Zilog

ADVANCED INFORMATION
Product Specification

Z8001®/Z8002®
Z8000® CPU
Central Processing Unit

October 1988

FEATURES

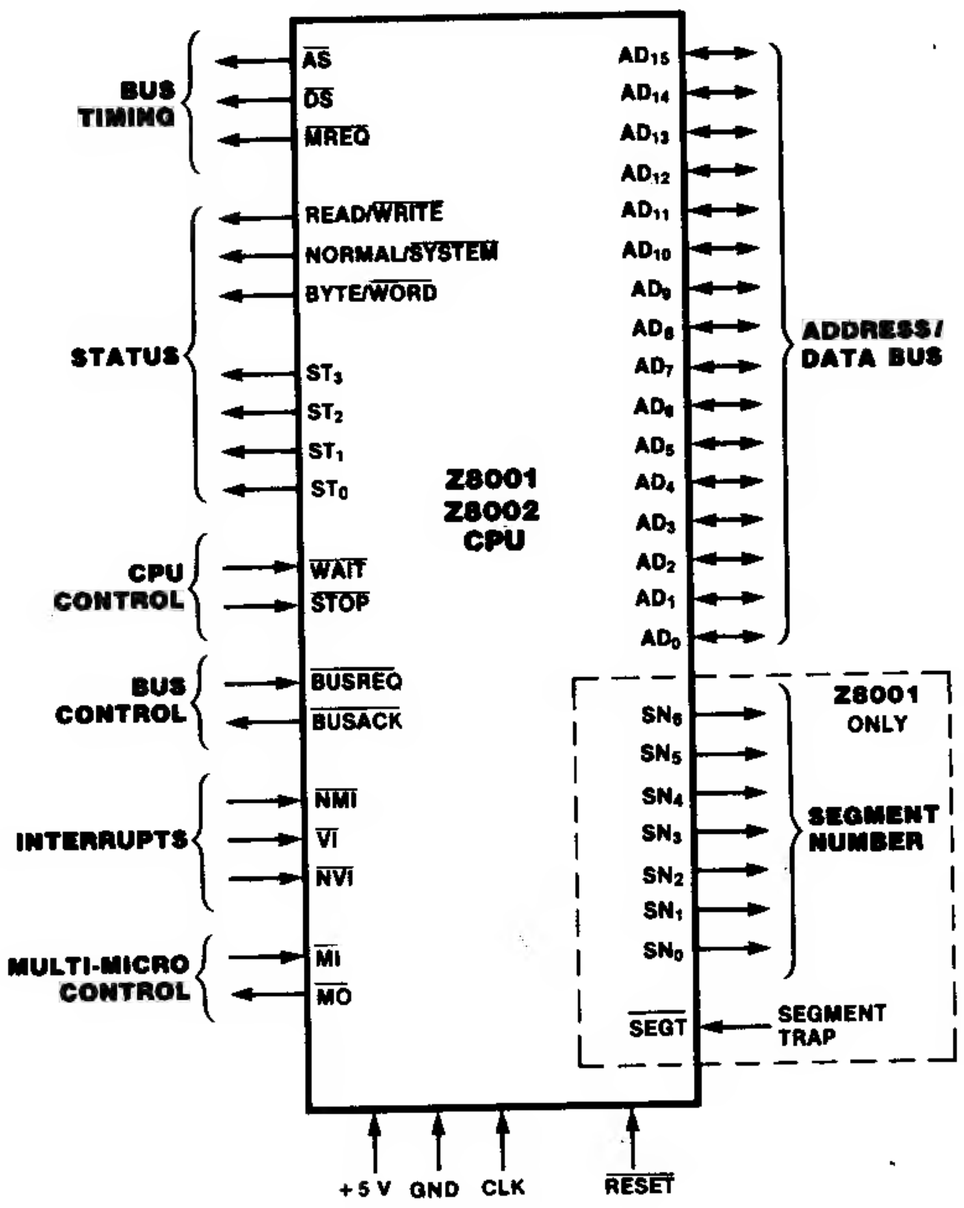
- Regular, easy-to-use architecture
- Instruction set more powerful than many minicomputers
- Directly addresses 8 Mbytes
- Eight user-selectable addressing modes
- Seven data types that range from bits to 32-bit long words and byte and word strings
- System and Normal operating modes
- Separate code, data, and stack spaces
- Sophisticated interrupt structure
- Resource-shaping capabilities for multiprocessing systems
- Multi-programming support
- Compiler support
- Memory management and protection provided by Z8010 Memory Management Unit
- 32-bit operations, including signed multiply and divide
- Z-BUS compatible
- 4, 6, and 10 MHz clock rate

GENERAL DESCRIPTION

The Z8000 is an advanced high-end 16-bit microprocessor that spans a wide variety of applications ranging from simple stand-alone computers to complex parallel-processing systems. Essentially a monolithic minicomputer central processing unit, the Z8000 CPU is characterized by an instruction set more powerful than many minicomputers; abundant resources in registers, data types, addressing modes and addressing range, and a regular architecture that enhances throughput by avoiding critical bottlenecks such as implied or dedicated registers.

CPU resources include sixteen 16-bit general-purpose registers, seven data types that range from bits to 32-bit long words and byte and word strings, and eight user-selectable addressing modes. The 110 distinct instruction types can be combined with the various data types and addressing modes to form a powerful set of 414 instructions. Moreover, the instruction set is regular; most instructions can use any of the five main addressing modes and can operate on byte, word, and long-word data types.

The CPU can operate in either the system or normal mode. The distinction between these two modes permits privileged operations, thereby improving operating system organization and implementation. Multiprogramming is supported by the "atomic" Test and Set instruction; multiprocessing by a combination of instruction and



hardware features; and compilers by multiple stacks, special instructions, and addressing modes.

The Z8000 CPU is offered in three versions: the Z8001/ Z160 segmented CPUs and the Z8002 nonsegmented CPU (Figure 1). The main difference is in addressing range. The Z8001 can directly address 8 megabytes of memory; the Z160 directly addresses 2 megabytes; the Z8002 directly addresses 64 kilobytes. The two operating modes - system and normal - and the distinction between code, data, and stack spaces within each mode allows memory extension up to 48 megabytes for the Z8001, 12 megabytes for the Z160 and 384 kilobytes for the Z8002.

To meet the requirements of complex, memory-intensive applications, a companion memory-management device is offered for the Z8001. The Z8010 Memory Management Unit manages the large address space by providing features such as segment relocation and memory protection. The Z8001 can be used with or without the Z8010. If used by itself, the Z8001 still provides an 8 megabyte direct addressing range, extendable to 48 megabytes.

The Z8001, Z8002 and Z8010 are fabricated with high-density, high-performance scaled n-channel silicon-gate depletion-load technology, and are housed in dual-in-line packages (DIPs) and leadless chip carriers (LCC).

REGISTER ORGANIZATION

The Z8000 CPU is a register-oriented machine that offers sixteen 16-bit general-purpose registers and a set of special system registers. All general-purpose registers can be used as accumulators and all but one as index registers or memory pointers.

Register flexibility is created by grouping and overlapping multiple registers (Figures 2 and 3). For byte operations, the first eight 16-bit registers (R0... R7) are treated as sixteen 8-bit registers (RL0, RH0..., RL7, RH7). The sixteen 16-bit registers are grouped in pairs (RR0... RR14) to form 32-bit long-word registers. Similarly, the register set is grouped in quadruples (RQ0... RQ12) to form 64-bit registers.

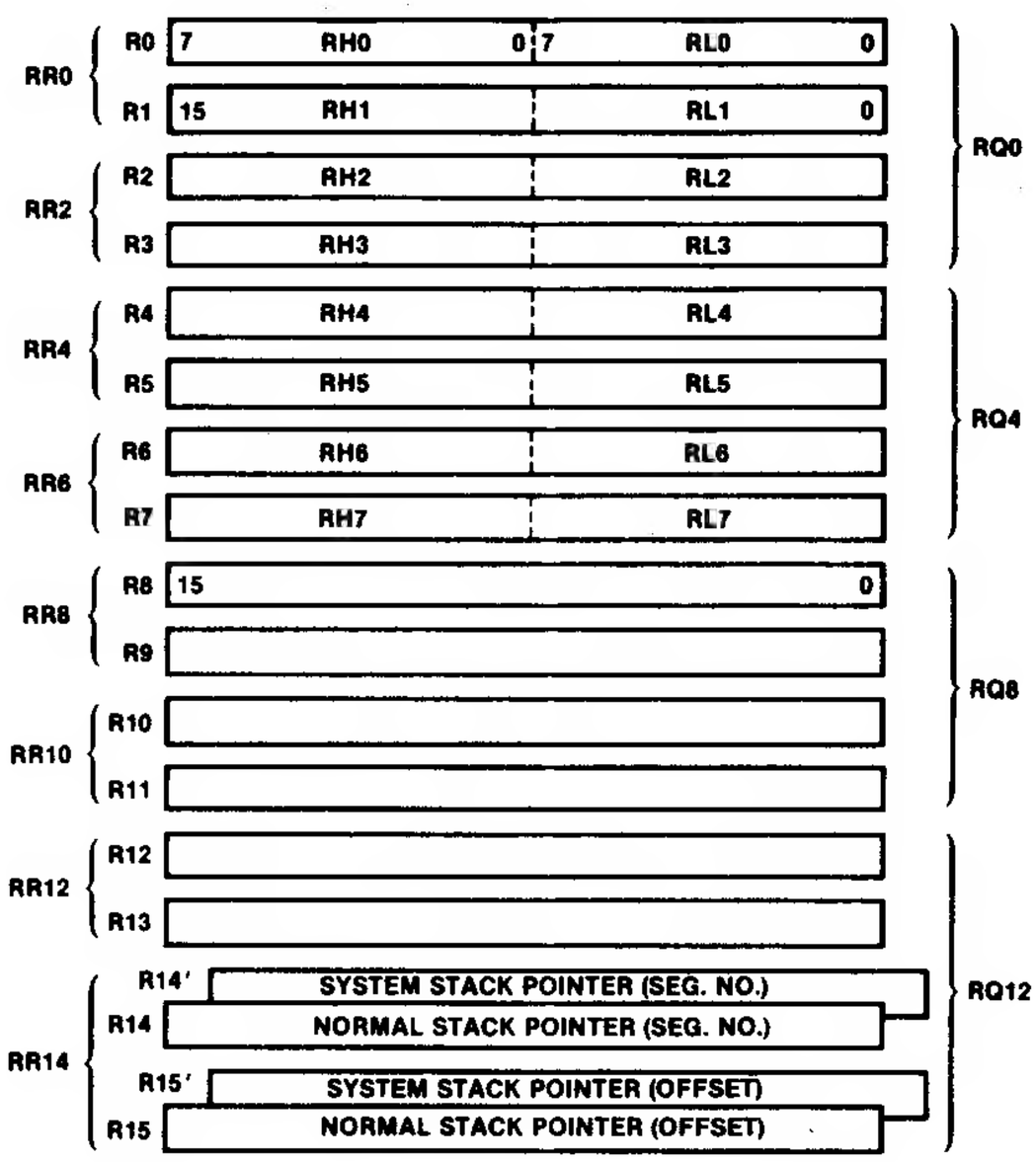


Figure 2. Z8001 General-Purpose Registers

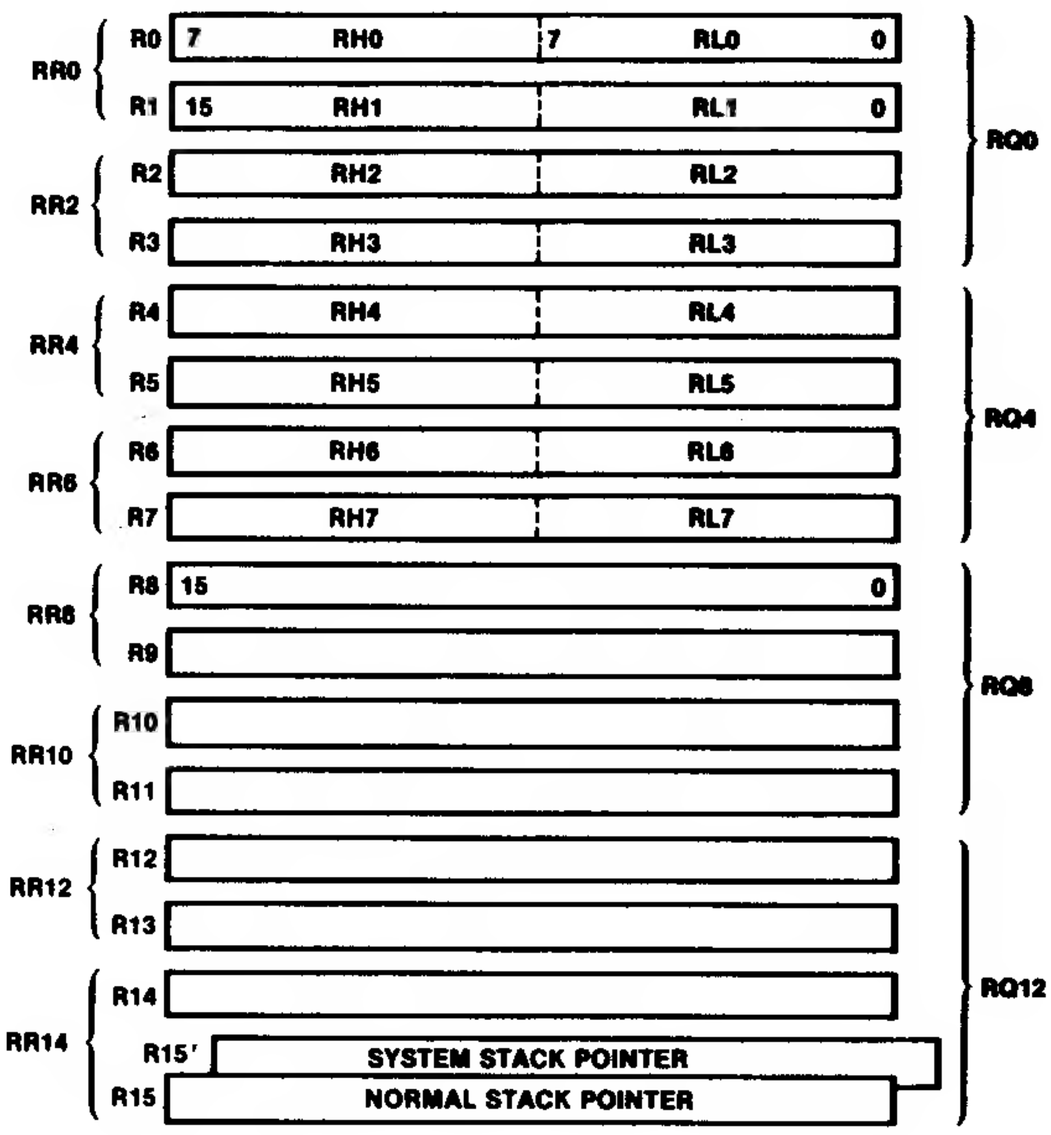


Figure 3. Z8002 General-Purpose Registers

STACKS

The Z8001, Z8002 and Z160 can use stacks located anywhere in memory. Call and Return instructions as well as interrupts and traps use implied stacks. The distinction between normal and system stacks separates system information from the application program information. Two stack pointers are available: the system stack pointer and the normal stack pointer. Because they are part of the general-purpose register group, the user can manipulate the stack pointers with any instruction available for register operations.

In the Z8001, register pair RR14 is the implied stack pointer. Register R14 contains the 7-bit segment number and R15 contains the 16-bit offset. In the Z8002, register R15 is the implied 16-bit stack pointer.

REFRESH

The Z8000 CPU contains a counter that can be used to automatically refresh dynamic memory. The refresh counter register consists of a 9-bit row counter, a 6-bit rate counter, and an enable bit (Figure 4). The 9-bit row counter can address up to 256 rows and is incremented by two each time the rate counter reaches end-of-count. The rate counter determines the time between successive refreshes. It consists of a programmable 6-bit modulo-n prescaler (n = 1 to 64), driven at one-fourth the CPU clock rate. The refresh period can be programmed by 1 to 64 μs with a 4 MHz clock. Refresh can be disabled by programming the refresh enable/disable bit.

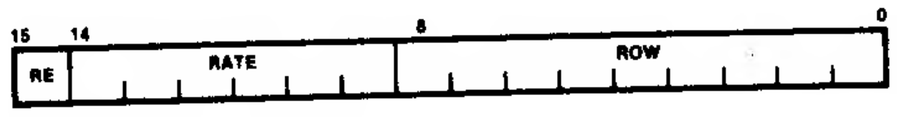


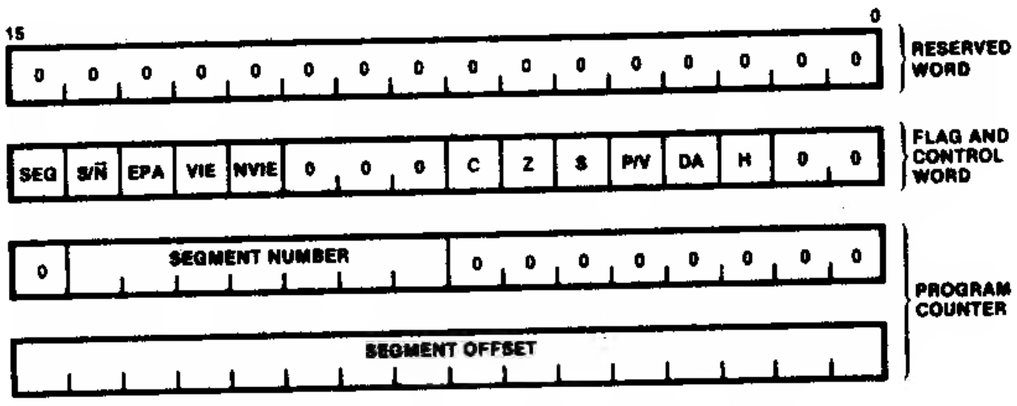
Figure 4. Refresh Counter

PROGRAM STATUS INFORMATION

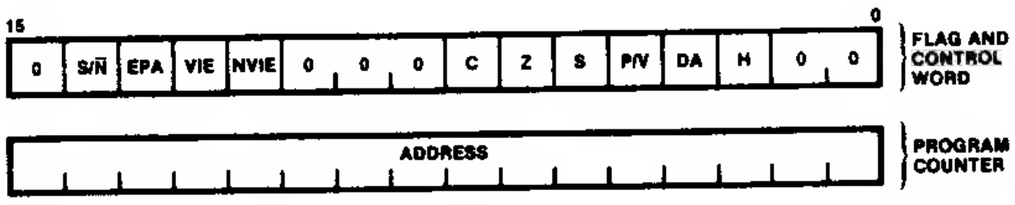
This group of status registers contains the program counter, flags, and control bits. When an interrupt or trap occurs, the entire group is saved and a new program status group is loaded.

Figure 5 illustrates how the program status groups of the Z8001 and Z8002 differ. In the nonsegmented Z8002, the program status group consists of two words: the program counter (PC), and the flag and control word (FCW). In the segmented Z8001, the program status group consists of four words: a two-word program counter, the flag and control word, and an unused word reserved for future use. Seven bits of the first PC word designate one of the 128 memory segments. The second word supplies the 16-bit offset that designates a memory location within the segment.

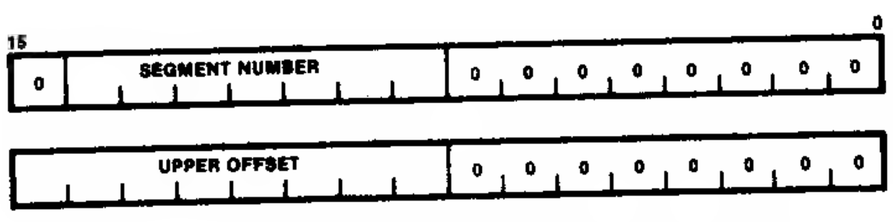
With the exception of the segment enable bit in the Z8001 program status group, the flags and control bits are the same for both CPUs.



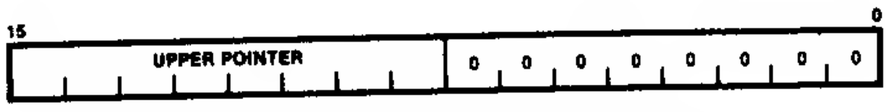
Z8001 Program Status Registers



Z8002 Program Status Registers



Z8001 Program Status Area Pointer



Z8002 Program Status Area Pointer

Figure 5. Z8000 CPU Special Registers

INTERRUPT AND TRAP STRUCTURE

The Z8000 provides a very flexible and powerful interrupt and trap structure. Interrupts are external asynchronous events requiring CPU attention, and are generally triggered by peripherals needing service. Traps are synchronous events resulting from the execution of certain instructions. Both are processed in a similar manner by the CPU.

The CPU supports three types of interrupts (non-maskable, vectored, and non-vectored) and four traps [system call, Extended Process Architecture (EPA) instruction, privileged instructions, and segmentation trap]. The vectored and non-vectored interrupts are maskable. Of the four traps, the only external one is the segmentation trap, which is generated by the Z8010.

The remaining traps occur when instructions limited to the system mode are used in the normal mode, or as a result of the System Call instruction, or for an EPA instruction. The descending order of priority for traps and interrupts is: internal traps, nonmaskable interrupt, segmentation trap, vectored interrupt, and non-vectored interrupt.

When an interrupt or trap occurs, the current program status is automatically pushed on the system stack. The program status consists of the processor status (PC and FCW) plus a 16-bit identifier. The identifier contains the reason or source of the trap or interrupt. For internal traps, the identifier is the first word of the trapped instruction. For external traps or interrupts, the identifier is the vector on the data bus read by the CPU during the interrupt-acknowledge or trap-acknowledge cycle.

After saving the current program status, the new program status is automatically loaded from the program status area in system memory. This area is designated by the program status area pointer (PSAP).

DATA TYPES

Z8000 instructions can operate on bits, BCD digits (4 bits), bytes (8 bits), words (16 bits), long words (32 bits), and byte strings and word strings (up to 64 kilobytes long). Bits can be set, reset, and tested; digits are used in BCD arithmetic operations; bytes are used for characters or small integer values; words are used for integer values, instructions and nonsegmented addresses; long words are used for long integer values and segmented addresses. All data elements except strings can reside either in registers or memory. Strings are stored in memory only.

The basic data element is the byte. The number of bytes used when manipulating a data element is either implied by the operation or—for strings and multiple register operations—explicitly specified in the instruction.

SEGMENTATION AND MEMORY MANAGEMENT

High-level languages, sophisticated operating systems, large programs and data bases, and decreasing memory prices are all accelerating the trend toward larger memory requirements in microcomputer systems. The Z8001 meets this requirement with an eight megabyte addressing space. This large address space is directly accessed by the CPU using a segmented addressing scheme and can be managed by the Z8010 Memory Management Unit.

Segmented Addressing

A segmented addressing space—compared with linear addressing—is closer to the way a programmer uses memory because each procedure and data space resides in its own segment. The 8 megabytes of Z8001 addressing space is divided into 128 relocatable segments up to 64 kilobytes each. A 23-bit segmented address uses a 7-bit segment address to point to the segment, and a 16-bit offset to address any location relative to the beginning of the segment. The two parts of the segmented address may be manipulated separately. The segmented Z8001 can run any code written for the nonsegmented Z8002 in any one of its 128 segments, provided it is set to the nonsegmented mode.

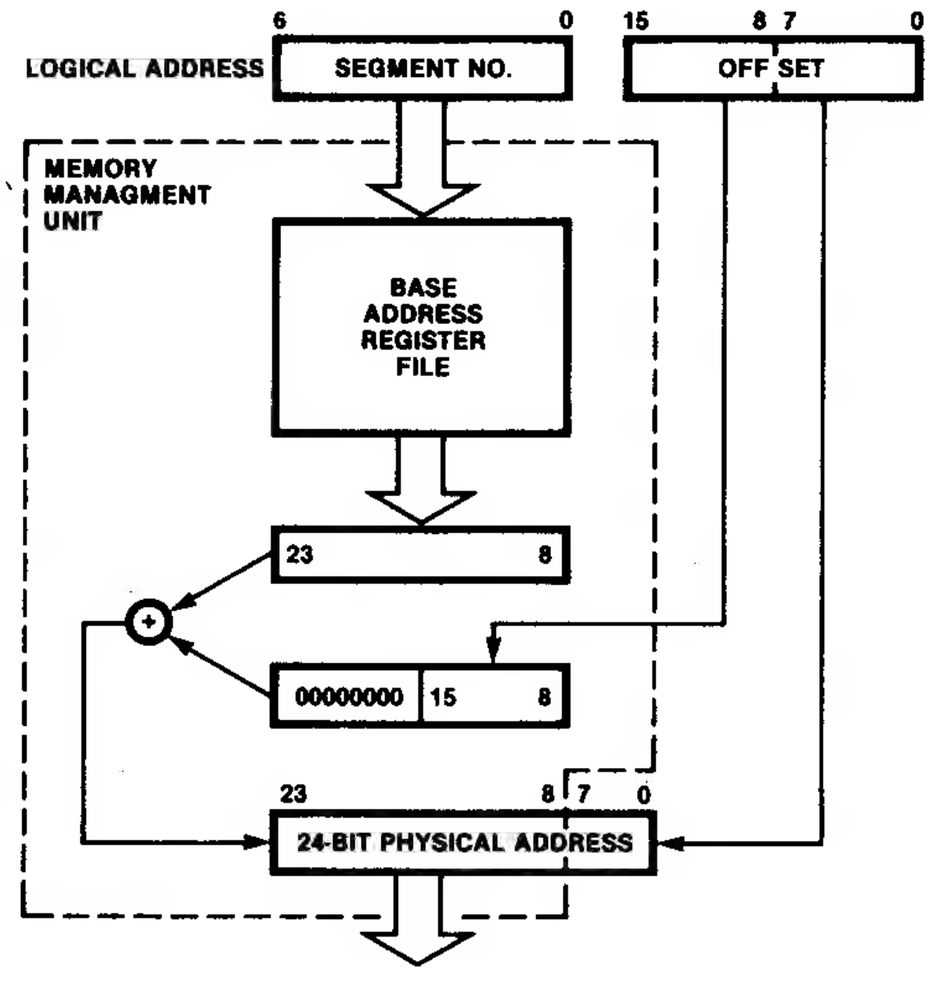


Figure 6. Logical-to-Physical Address Translation

In hardware, segmented addresses are contained in a register pair or long-word memory location. The segment number and offset can be manipulated separately or together by all the available word and long-word operations.

When contained in an instruction, a segmented address has two different representations: long offset and short offset. The long offset occupies two words, whereas the short offset requires only one and combines in one word the 7-bit segment number with an 8-bit offset (range 0-256). The short offset mode allows very dense encoding of addresses and minimizes the need for long addresses required by direct accessing of this large address space.

Memory Management

The addresses manipulated by the programmer, used by instructions and output by the Z8001, are called *logical* addresses. The Memory Management Unit takes the logical addresses and transforms them into the *physical* addresses required for accessing the memory (Figure 6). This address transformation process is called relocation. Segment relocation makes user software addresses independent of the physical memory so the user is freed from specifying where information is actually located in the physical memory.

The relocation process is transparent to user software. A translation table in the Memory Management Unit associates the 7-bit segment number with the base address of the physical memory segment. The 16-bit offset is added to the physical base address to obtain the actual physical address. The system may dynamically reload translation tables as tasks are created, suspended, or changed.

In addition to supporting dynamic segment relocation, the Memory Management Unit also provides segment protection and other segment management features. The protection features prevent illegal uses of segments, such as writing into a write-protected zone.

Each Memory Management Unit stores 64 segment entries that consist of the segment base address, its attributes, size, and status. Segments are variable in size from 256 bytes to 64 kilobytes in increments of 256 bytes. Pairs of Management Units support the 128 segment numbers available for each of the six CPU address spaces. Within an address space, several Management Units can be used to create multiple translation tables.

EXTENDED PROCESSING ARCHITECTURE

The Zilog Extended Processing Architecture (EPA) provides an extremely flexible and modular approach to expanding both the hardware and software capabilities of the Z8000 CPU. Features of the EPA include:

- Specialized instructions for external processors or software traps may be added to CPU instruction set.
- Increases throughput of the system by using up to four specialized external processors in parallel with the CPU.
- Permits modular design of Z8000-based systems.
- Provides easy management of multiple microprocessor configurations via "single instruction stream" communication.
- Simple interconnection between extended processing units and Z8000 CPU requires no additional external supporting logic.
- Supports debugging of suspect hardware against proven software.
- Standard features on all Zilog Z8000 CPUs.

Specific benefits include:

- EPUs can be added as the system grows and as EPUs with specialized functions are developed.
- Control of EPUs is accomplished via a "single instruction stream" in the Z8000 CPU, eliminating many significant system software and bus contention management obstacles that occur in other multiprocessor (e.g., master-slave) organization schemes.

The processing power of the Zilog Z8000 16-bit microprocessor can be boosted beyond its intrinsic capability by Extended Processing Architecture. Simply stated, EPA allows the Z8000 CPU to accommodate up to four Extended Processing Units (EPUs), which perform specialized functions in parallel with the CPU's main instruction execution stream (Figure 7).

The use of extended processors to boost the main CPU's performance capability has been proven with large mainframe computers and minicomputers. In these systems, specialized functions such as array processing, special input/output processing, and data communications processing are typically assigned to extended processor hardware. These extended processors are complex computers in their own right.

The Zilog Extended Processing Architecture combines the best concepts of these proven performance boosters with the latest in high-density MOS integrated-circuit design. The result is an elegant expansion of design capability—a powerful microprocessor architecture capable of connecting single-chip EPUs that permits very effective parallel processing and makes for a smoothly integrated instruction stream from the Z8000 programmer's point of view. A typical addition to the current Z8000 instruction set is a set of Floating Point Instructions.

The Extended Processing Units connect directly to the Z8000 Bus (Z-BUS) and continuously monitor the CPU instruction stream. When an extended instruction is detected, the appropriate EPU responds, obtaining or

placing data or status information on the Z-BUS using the Z8000-generated control signals and performing its function as directed.

The Z8000 CPU is responsible for instructing the EPU and delivering operands and data to it. The EPU recognizes instructions intended for it and executes them, using data supplied with the instruction and/or data within its internal registers. There are four classes of EPU instructions:

- Data transfers between main memory and EPU registers
- Data transfers between CPU registers and EPU registers
- EPU internal operations
- Status transfers between the EPUs and the Z8000 CPU Flag and Control Word register (FCW)

Four Z8000 addressing modes may be utilized with transfers between EPU registers and the CPU and main memory:

- Register
- Indirect Register
- Direct Address
- Index

In addition to the hardware-implemented capabilities of the Extended Processing Architecture, there is an extended instruction trap mechanism to permit software simulation of EPU functions. A control bit in the Z8000 FCW register indicates whether actual EPUs are present or not. If not, when an extended instruction is detected, the Z8000 traps on the instruction, so that a software "trap handler" can emulate the desired EPU function—a very useful development tool. The EPA software trap routine supports the debugging of suspect hardware against proven software. This feature will increase in significance as designers become familiar with the EPA capability of the Z8000 CPU.

This software trap mechanism facilitates the design of systems for later addition of EPUs: initially, the extended function is executed as a trap subroutine; when the EPU is finally attached, the trap subroutine is eliminated and the EPA control bit is set. Application software is unaware of the change.

Extended Processing Architecture also offers protection against extended instruction overlapping. Each EPU connects to the Z8000 CPU via the STOP line so that if an EPU is requested to perform a second extended instruction function before it has completed the previous one, it can put the CPU into the Stop/Refresh state until execution of the previous extended instruction is complete.

EPA and CPU instruction execution are shown in Figure 8. The CPU begins operation by fetching an instruction and determining whether it is a CPU or an EPU command. The EPU meanwhile monitors the Z-BUS for its own instructions. If the CPU encounters an EPU command, it checks to see whether an EPU is present; if not, the EPU may be simulated by an EPU instruction trap software routine; if an EPU is present, the necessary data and/or address is placed on the Z-BUS. If the EPU is free when the instruction and data for it appear, the extended instruction is executed. If the EPU is still processing a previous instruction, it activates the CPU's STOP line to lock the CPU off at the Z-BUS until execution is complete. After the instruction is finished, the EPU deactivates the STOP line and CPU transactions continue.

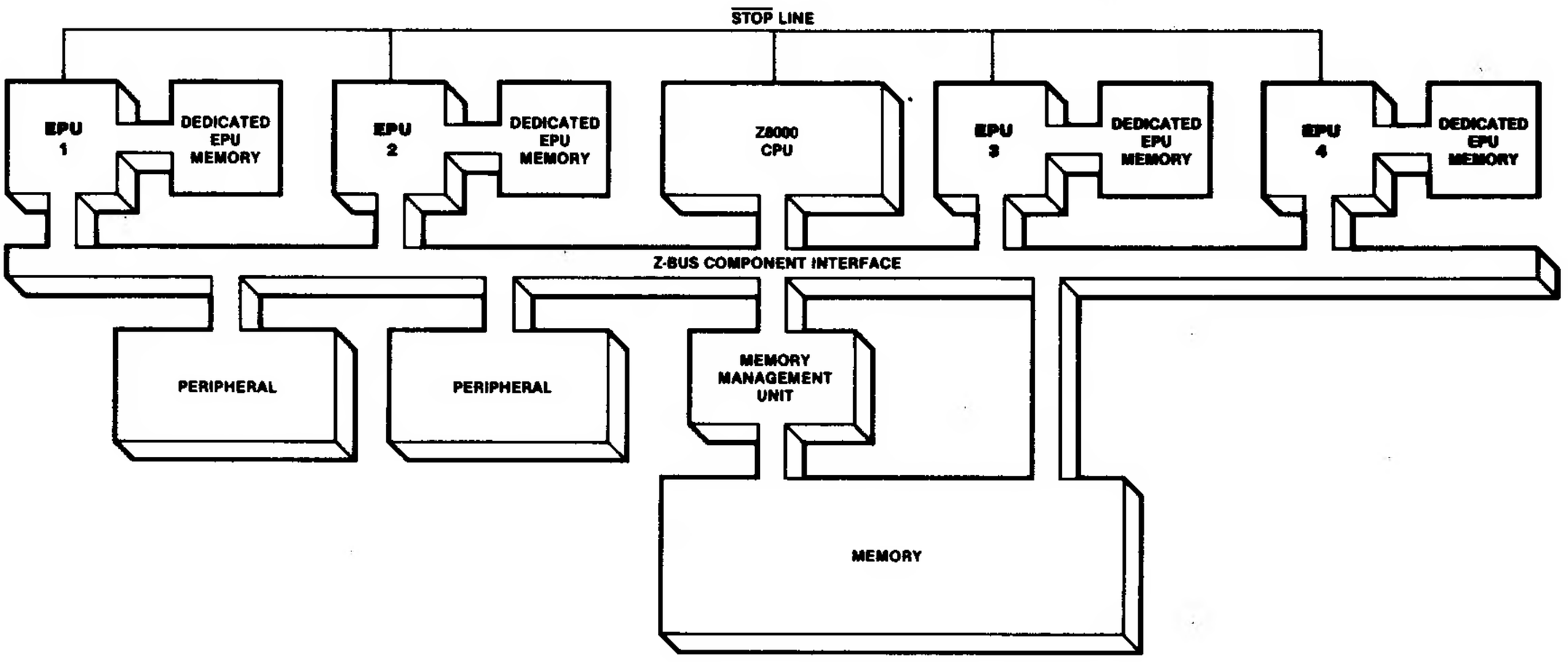
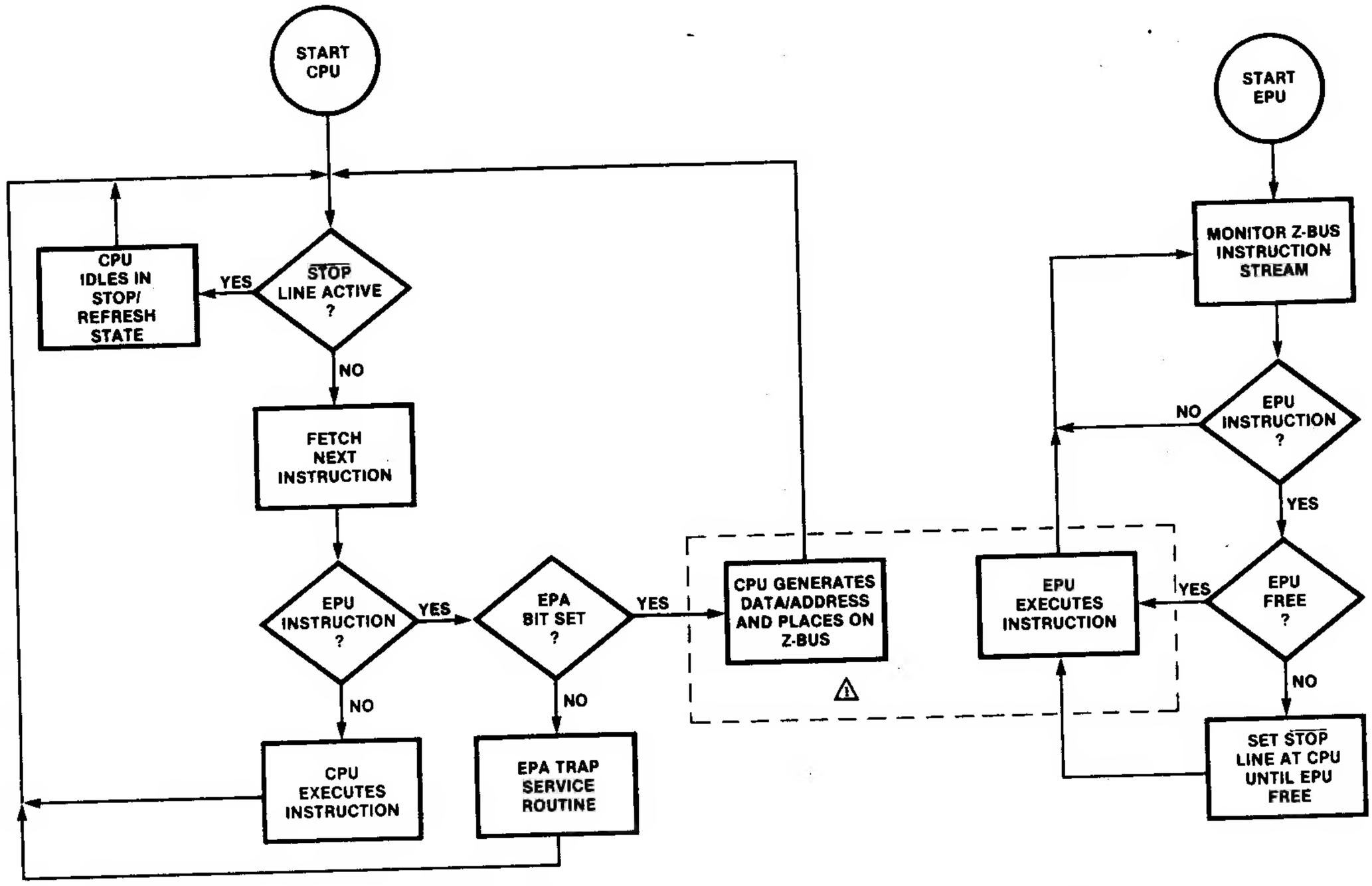


Figure 7. Typical Extended Processor Configuration



1 DATA OR ADDRESSES ARE PLACED ON THE BUS AND USED BY THE EPU IN THE EXECUTION OF AN INSTRUCTION.

Figure 8. EPA and Z8000 CPU Instruction Execution

INPUT/OUTPUT

A set of I/O instructions performs 8-bit or 16-bit transfers between the CPU and I/O devices. I/O devices are addressed with a 16-bit I/O port address. The I/O port address is similar to a memory address; however, I/O address space need not be part of the memory address space. I/O port and memory addresses coexist on the same bus lines and they are distinguished by the status outputs.

Two types of I/O instructions are available: standard and special. Each has its own address space. The I/O instructions include a comprehensive set of In, Out, and Block I/O instructions for both bytes and words. Special I/O instructions are used for loading and unloading the Memory Management Unit. The status information distinguishes between standard and special I/O references.

MULTI-MICROPROCESSOR SUPPORT

Multi-microprocessor systems are supported in hardware and software. A pair of CPU pins is used in conjunction with certain instructions to coordinate multiple microprocessors. The Multi-Micro Out pin issues a request for the resource, while the Multi-Micro In pin is used to recognize the state of the resource. Thus, any CPU in a multiple microprocessor system can exclude all other asynchronous CPUs from a critical shared resource.

Multi-microprocessor systems are supported in software by the instructions Multi-Micro Request, Test Multi-Micro In, Set Multi-Micro Out, and Reset Multi-Micro Out. In addition, the eight megabyte CPU address space is beneficial in multiple microprocessor systems that have large memory requirements.

ADDRESSING MODES

The information included in Z8000 instructions consists of the function to be performed, the type and size of data elements to be manipulated, and the location of the data elements. Locations are designated by register addresses, memory addresses, or I/O addresses. The addressing mode of a given instruction defines the address space it references and the method used to compute the address itself. Addressing modes are explicitly specified or implied by the instruction.

Figure 9 illustrates the eight addressing modes: Register (R), Immediate (IM), Indirect Register (IR), Direct Address (DA), Index (X), Relative Address (RA), Base Address (BA), and Base Index (BX). In general, an addressing mode explicitly specifies either register address space or memory address space. Program memory address space and I/O address space are usually implied by the instruction.

Addressing Mode	Operand Addressing			Operand Value
	In the Instruction	In a Register	In Memory	
R				
Register		REGISTER ADDRESS → OPERAND		The content of the register
IM				
Immediate	OPERAND			In the instruction
***IR**				
Indirect Register		REGISTER ADDRESS → ADDRESS → OPERAND		The content of the location whose address is in the register
DA				
Direct Address		ADDRESS → OPERAND		The content of the location whose address is in the instruction
***X**				
Index		REGISTER ADDRESS + BASE ADDRESS → INDEX → OPERAND		The content of the location whose address is the address in the instruction plus the content of the working register.
RA				
Relative Address		DISPLACEMENT + PC VALUE → OPERAND		The content of the location whose address is the content of the program counter, offset by the displacement in the instruction
***BA**				
Base Address		REGISTER ADDRESS + DISPLACEMENT → BASE ADDRESS → OPERAND		The content of the location whose address is the address in the register, offset by the displacement in the instruction
***BX**				
Base Index		REGISTER ADDRESS + REGISTER ADDRESS → BASE ADDRESS + INDEX → OPERAND		The content of the location whose address is the address in a register plus the index value in another register.

*Do not use R0 or RR0 as indirect, index, or base registers.

Figure 9. Addressing Modes

INSTRUCTION SET SUMMARY

The Z8000 provides the following types of instructions:

- Load and Exchange
- Arithmetic
- Logical
- Program Control
- Bit Manipulation
- Rotate and Shift
- Block Transfer and String Manipulation
- Input/Output
- CPU Control

LOAD AND EXCHANGE

Mnemonics	Operands	Addr. Modes	Clock Cycles* Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
CLR	dst	R	7	7	7				**Clear**
CLRB		IR	8	8	8				dst ← 0
		DA	11	12	14				
		X	12	12	15				
EX	R, src	R	6	6	6				**Exchange**
EXB		IR	12	12	12				R ↔ src
		DA	15	16	18				
		X	16	16	19				
LD	R, src	R	3	3	3	5	5	5	**Load into Register**
LDB		IM	7	7	7	11	11	11	R ← src
LDL		IM	5 (byte only)						
		IR	7	7	7	11	11	11	
		DA	9	10	12	12	13	15	
		X	10	10	13	13	13	16	
		BA	14	14	14	17	17	17	
		BX	14	14	14	17	17	17	
LD	dst, R	IR	8	8	8	11	11	11	**Load into Memory** (Store)
LDB		DA	11	12	14	14	15	17	dst ← R
LDL		X	12	12	15	15	15	18	
		BA	14	14	14	17	17	17	
		BX	14	14	14	17	17	11	
LD	dst, IM	IR	11	11	11				**Load Immediate into Memory**
LDB		DA	14	15	17				dst ← IM
		X	15	15	18				
LDA	R, src	DA	12	13	15				**Load Address**
		X	13	13	16				R ← source address
		BA	15	15	15				
		BX	15	15	15				
LDAR	R, src	RA	15	15	15				**Load Address Relative**
									R ← source address
LDK	R, src	IM	5	5	5				**Load Constant**
									R ← n (n = 0... 15)
LDM	R, src, n	IR	11	11	11 + 3n				**Load Multiple**
		DA	14	15	17 + 3n				R ← src (n consecutive words)
		X	15	15	18 + 3n				(n = 1... 16)

*NS = Non-segmented SS = Segmented Short Offset SL = Segmented Long Offset

LOAD AND EXCHANGE (Continued)

Mnemonics	Operands	Addr. Modes	Clock Cycles* Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
LDM	dst, R, n	IR	11	11	11 + 3n				**Load Multiple** (Store Multiple)
		DA	14	15	17 + 3n				dst ← R (n consecutive words)
		X	15	15	18 + 3n				(n = 1... 16)
LDR	R, src	RA	14	14	14	17	17	17	**Load Relative**
LDRB									R ← src
LDRL									(range −32768... +32767)
LDR	dst, R	RA	14	14	14	17	17	17	**Load Relative** (Store Relative)
LDRB									dst ← R
LDRL									(range −32768... +32767)
POP	dst, IR	R	8	8	8	12	12	12	**Pop**
POPL		IR	12	12	12	19	19	19	dst ← IR
		DA	16	16	18	23	23	25	Autoincrement contents of R
		X	16	16	19	23	23	26	
PUSH	IR, src	R	9	9	9	12	12	12	**Push**
PUSHL		IM	12	12	12	19	19	19	Autodecrement contents of R
		IR	13	13	13	20	20	20	IR ← src
		DA	14	14	16	21	21	23	
		X	14	14	17	21	21	24	

ARITHMETIC

ADC	R, src	R	5	5	5				**Add with Carry**
ADCB									R ← R + src + carry
ADD	R, src	R	4	4	4	8	8	8	**Add**
ADDB		IM	7	7	7	14	14	14	R ← R + src
ADDL		IR	7	7	7	14	14	14	
		DA	9	10	12	15	16	18	
		X	10	10	13	16	16	19	
CP	R, src	R	4	4	4	8	8	8	**Compare with Register**
CPB		IM	7	7	7	14	14	14	R − src
CPL		IR	7	7	7	14	14	14	
		DA	9	10	12	15	16	18	
		X	10	10	13	16	16	19	
CP	dst, IM	IR	11	11	11				**Compare with Immediate**
CPB		DA	14	15	17				dst − IM
		X	15	15	18				
DAB	dst	R	5	5	5				**Decimal Adjust**
DEC	dst, n	R	4	4	4				**Decremented by n**
DECB		IR	11	11	11				dst ← dst − n
		DA	13	14	16				(n = 1... 16)
		X	14	14	17				

*NS = Non-segmented SS = Segmented Short Offset SL = Segmented Long Offset

ARITHMETIC (Continued)

Mnemonics	Operands	Addr. Modes	Clock Cycles* Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
DIV **DIVL**	R, src	R	107	107	107	744	744	744	**Divide** (signed) Word: $R_{n+1} \leftarrow R_{n,n+1}$ + src $R_n \leftarrow$ remainder Long Word: $R_{n+2,n+3} \leftarrow R_{n\ldots n+3}$ + src $R_{n,n+2} \leftarrow$ remainder
		IM	107	107	107	744	744	744	
		IR	107	107	107	744	744	744	
		DA	108	109	111	745	746	748	
		X	109	109	112	746	746	749	
EXTS **EXTSB** **EXTSL**	dst	R	11	11	11	11	11	11	**Extend Sign** Extend sign of low order half of dst through high order half of dst
INC **INCB**	dst, n	R	4	4	4				**Increment by n** dst ← dst + n (n = 1... 16)
		IR	11	11	11				
		DA	13	14	16				
		X	14	14	17				
MULT **MULTL**	R, src	R	70	70	70	282†	282†	282†	**Multiply** (signed) Word: $R_{n,n+1} \leftarrow R_{n+1} \bullet$ src Long Word: $R_{n\ldots n+3} \leftarrow R_{n+2,n+3}$ †Plus seven cycles for each 1 in the multiplicand
		IM	70	70	70	282†	282†	282†	
		IR	70	70	70	282†	282†	282†	
		DA	71	72	74	283†	284†	286†	
		X	72	72	75	284†	284†	287†	
NEG **NEGB**	dst	R	7	7	7				**Negate** dst ← 0 − dst
		IR	12	12	12				
		DA	15	16	18				
		X	16	16	19				
SBC **SBCB**	R, src	R	5	5	5				**Subtract with Carry** R ← R − src − carry
SUB **SUBB** **SUBL**	R, src	R	4	4	4	8	8	8	**Subtract** R ← R − src
		IM	7	7	7	14	14	14	
		IR	7	7	7	14	14	14	
		DA	9	10	12	15	16	18	
		X	10	10	13	16	16	19	

LOGICAL

AND **ANDB**	R, src	R	4	4	4				**AND** R ← R AND src
		IM	7	7	7				
		IR	7	7	7				
		DA	9	10	12				
		X	10	10	13				
COM **COMB**	dst	R	7	7	7				**Complement** dst ← NOT dst
		IR	12	12	12				
		DA	15	16	18				
		X	16	16	19				
OR **ORB**	R, src	R	4	4	4				**OR** R ← R OR src
		IM	7	7	7				
		IR	7	7	7				
		DA	9	10	12				
		X	10	10	13				

*NS = Non-segmented SS = Segmented Short Offset SL = Segmented Long Offset

LOGICAL (Continued)

Mnemonics	Operands	Addr. Modes	Clock Cycles* Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
TCC	cc, dst	R	5	5	5				**Test Condition Code**
TCCB									Set LSB if cc is true
TEST	dst	R	7	7	7	13	13	13	**Test**
TESTB		IR	8	8	8	13	13	13	dst OR 0
TESTL		DA	11	12	14	16	17	19	
		X	12	12	15	17	17	20	
XOR	R, src	R	4	4	4				**Exclusive OR**
XORB		IM	7	7	7				R ← R XOR src
		IR	7	7	7				
		DA	9	10	12				
		X	10	10	13				

PROGRAM CONTROL

CALL	dst	IR	10	15	15		**Call Subroutine**
		DA	12	18	20		Autodecrement SP
		X	13	18	21		@ SP ← PC
							PC ← dst
CALR	dst	RA	10	10	15		**Call Relative**
							Autodecrement SP
							@ SP ← PC
							PC←PC + dst (range – 4094 to + 4096)
DJNZ	R, dst	RA	11	11	11		**Decrement and Jump if Non-Zero**
DBJNZ							R ← R – 1
							If R ≠ 0: PC←PC + dst(range – 254 to 9)
IRET†	—	—	13	13	16		**Interrupt Return**
							PS ← @ SP
							Autoincrement SP
JP	cc, dst	IR	10	10	15	(taken)	**Jump Conditional**
		IR	7	7	7	(not taken)	If cc is true: PC ← dst
		DA	7	8	10		
		X	8	8	11		
JR	cc, dst	RA	6	6	6		**Jump Conditional Relative**
							If cc is true: PC ← PC + dst
							(range – 256 to + 254)
RET	cc	—	10	10	13	(taken)	**Return Conditional**
			7	7	7	(not taken)	If cc is true: PC ← @ SP
							Autoincrement SP
SC	src	IM	33	33	39		**System Call**
							Autodecrement SP
							@ SP ← old PS
							Push instruction
							PS ← System Call PS

*NS = Non-segmented SS = Segmented Short Offset SL = Segmented Long Offset
†Privileged instruction Executed in system mode only.

BIT MANIPULATION

Mnemonics	Operands	Addr. Modes	Clock Cycles* Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
BIT **BITB**	dst, b	R	4	4	4				**Test Bit Static** Z flag ← NOT dst bit specified by b
		IR	8	8	8				
		DA	10	11	13				
		X	11	11	14				
BIT **BITB**	dst, R	R	10	10	10				**Test Bit Dynamic** Z flag ← NOT dst bit specified by contents of R
RES **RESB**	dst, b	R	4	4	4				**Reset Bit Static** Reset dst bit specified by b
		IR	11	11	11				
		DA	13	14	16				
		X	14	14	17				
RES **RESB**	dst, R	R	10	10	10				**Reset Bit Dynamic** Reset dst bit specified by contents R
SET **SETB**	dst, b	R	4	4	4				**Set Bit Static** Set dst bit specified by b
		IR	11	11	11				
		DA	13	14	16				
		X	14	14	17				
SET **SETB**	dst, R	R	10	10	10				**Set Bit Dynamic** Set dst bit specified by contents of R
TSET **TSETB**	dst	R	7	7	7				**Test and Set** S flag ← MSB of dst dst ← all 1s
		IR	11	11	11				
		DA	14	15	17				
		X	15	15	18				

ROTATE AND SHIFT

RL **RLB**	dst, n	R	6 for n = 1						**Rotate Left** by n bits (n = 1, 2)
		R	7 for n = 2						
RLC **RLCB**	dst, n	R	6 for n = 1						**Rotate Left through Carry** by n bits (n = 1, 2)
		R	7 for n = 2						
RLDB	R, src	R	9	9	9				**Rotate Digit Left**
RR **RRB**	dst, n	R	6 for n = 1						**Rotate Right** by n bits (n = 1, 2)
		R	7 for n = 2						
RRC **RRCB**	dst, n	R	6 for n = 1						**Rotate Right through Carry** by n bits (n = 1, 2)
		R	7 for n = 2						
RRDB	R, src	R	9	9	9				**Rotate Digit Right**
SDA **SDAB** **SDAL**	dst, R	R	(15 + 3 n)			(15 + 3 n)			**Shift Dynamic Arithmetic** Shift dst left or right by contents of R
SDL **SDLB** **SDLL**	dst, R	R	(15 + 3 n)			(15 + 3 n)			**Shift Dynamic Logical** Shift dst left or right by contents of R

*NS = Non-segmented SS = Segmented Short Offset SL = Segmented Long Offset

ROTATE AND SHIFT (Continued)

Mnemonics	Operands	Addr. Modes	Clock Cycles* Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
SLA **SLAB** **SLAL**	dst, n	R	(13 + 3 n)			(13 + 3 n)			**Shift Left Arithmetic** by n bits
SLL **SLLB** **SLLL**	dst, n	R	(13 + 3 n)			(13 + 3 n)			**Shift Left Logical** by n bits
SRA **SRAB** **SRAL**	dst, n	R	(13 + 3 n)			(13 + 3 n)			**Shift Right Arithmetic** by n bits
SRL **SRLB** **SRLL**	dst, n	R	(13 + 3 n)			(13 + 3 n)			**Shift Right Logical** by n bits

BLOCK TRANSFER AND STRING MANIPULATION

CPD **CPDB**	R_X,src,R_Y,cc	IR	20	20	20				**Compare and Decrement** R_X – src Autodecrement src address $R_Y \leftarrow R_Y - 1$
CPDR **CPDRB**	R_X,src,R_Y,cc	IR	(11 + 9 n)						**Compare, Decrement, and Repeat** R_X – src Autodecrement src address $R_Y \leftarrow R_Y - 1$ Repeat until cc is true or $R_Y = 0$
CPI **CPIB**	R_X,src,R_Y,cc	IR	20	20	20				**Compare and Increment** R_X – src Autoincrement src address $R_Y \leftarrow R_Y - 1$
CPIR **CPIRB**	R_X,src,R_Y,cc	IR	(11 + 9 n)						**Compare, Increment, and Repeat** R_X – src Autoincrement src address $R_Y \leftarrow R_Y - 1$ Repeat until cc is true or $R_Y = 0$
CPSD **CPSDB**	dst,src,R,cc	IR	25	25	25				**Compare String and Decrement** dst – src Autodecrement dst and src addresses R ← R – 1
CPSDR **CPSDRB**	dst,src,R,cc	IR	(11 + 14 n)						**Compare String, Decrement, and Repeat** dst – src Autodecrement dst and src addresses R ← R – 1 Repeat until cc is true or R = 0

*NS = Non-segmented SS = Segmented Short Offset SL = Segmented Long Offset

BLOCK TRANSFER AND STRING MANIPULATION (Continued)

Mnemonics	Operands	Addr. Modes	Clock Cycles* Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
CPSI **CPSIB**	dst,src,R,cc	IR	25	25	25				**Compare String and Increment** dst – src Autoincrement dst and src addresses R ← R – 1
CPSIR **CPSIRB**	dst,src,R,cc	IR	(11 + 14 n)						**Compare String, Increment and Repeat** dst – src Autoincrement dst and src addresses R ← R – 1 Repeat until cc is true or R = 0
LDD **LDDB**	dst,src,R	IR	20	20	20				**Load and Decrement** dst ← src Autodecrement dst and src addresses R ← R – 1
LDDR **LDDRB**	dst,src,R	IR	(11 + 9 n)						**Load, Decrement and Repeat** dst ← src Autodecrement dst and src addresses R ← R – 1 Repeat until R = 0
LDI **LDIB**	dst,src,R	IR	20	20	20				**Load and Increment** dst ← src Autoincrement dst and src addresses R ← R – 1
LDIR **LDIRB**	dst,src,R	IR	(11 + 9 n)						**Load, Increment and Repeat** dst ← src Autoincrement dst and src addresses R ← R – 1 Repeat until R = 0
TRDB	dst,src,R	IR	25	25	25				**Translate and Decrement** dst ← src (dst) Autodecrement dst address R ← R – 1
TRDRB	dst,src,R	IR	(11 + 14 n)						**Translate, Decrement and Repeat** dst ← src (dst) Autodecrement dst address R ← R – 1 Repeat until R = 0
TRIB	dst,src,R	IR	25	25	25				**Translate and Increment** dst ← src (dst) Autoincrement dst address R ← R – 1

*NS = Non-segmented SS = Segmented Short Offset SL = Segmented Long Offset
*Privileged instruction. Executed in system mode only.

BLOCK TRANSFER AND STRING MANIPULATION (Continued)

Mnemonics	Operands	Addr. Modes	Clock Cycles* Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
TRIRB	dst,src,R	IR	(11 + 14 n)						**Translate, Increment and Repeat** dst ← src (dst) Autoincrement dst address R ← R − 1 Repeat until R = 0
TRTDB	src1,src2,R	IR	25	25	25				**Translate and Test, Decrement** RH1 ← src2 (src1) Autodecrement src 1 address R ← R − 1
TRTDRB	src1,src2,R	IR	(11 + 14 n)						**Translate and Test, Decrement, and Repeat** RH1 ← src2 (src1) Autodecrement src1 address R ← R − 1 Repeat until R = 0 or RH1 = 0
TRTIB	src1,src2,R	IR	25	25	25				**Translate and Test, Increment** RH1 ← src2 (src1) Autoincrement src1 address R ← R − 1
TRTIRB	src1,src2,R	IR	(11 + 14 n)						**Translate and Test, Increment and Repeat** RH1 ← src2 (src1) Autoincrement src 1 address R ← R − 1 Repeat until R = 0 or RH1 = 0

INPUT/OUTPUT

IN†	R,src	IR	10	10	10				**Input** R ← src
INB†		DA	12	12	12				
IND† **INDB**†	dst,src,R	IR	21	21	21				**Input and Decrement** dst ← src Autodecrement dst address R ← R − 1
INDR† **INDRB**†	dst,src,R	IR	(11 + 10 n)						**Input, Decrement and Repeat** dst ← src Autodecrement dst address R ← R − 1 Repeat until R = 0
INI† **INIB**†	dst,src,R	IR	21	21	21				**Input and Increment** dst ← src Autoincrement dst address R ← R − 1

*NS = Non-segmented SS = Segmented Short Offset SL = Segmented Long Offset
†Privileged instruction Executed in system mode only

INPUT/OUTPUT (Continued)

Mnemonics	Operands	Addr. Modes	Clock Cycles* Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
INIR† **INIRB**†	dst,src,R	IR	(11 + 10 n)						**Input, Increment and Repeat** dst ← src Autoincrement dst address R ← R − 1 Repeat until R = 0
OUT† **OUTB**†	dst,R	IR	10	10	10				**Output** dst ← R
		DA	12	12	12				
OUTD† **OUTDB**†	dst,src,R	IR	21	21	21				**Output and Decrement** dst ← src Autodecrement src address R ← R − 1
OTDR† **OTDRB**†	dst,src,R	IR	(11 + 10 n)						**Output, Decrement and Repeat** dst ← src Autodecrement src address R ← R − 1 Repeat until R = 0
OUTI† **OUTIB**†	dst,src,R	IR	21	21	21				**Output and Increment** dst ← src Autoincrement src address R ← R − 1
OTIR† **OTIRB**†	dst,src,R	IR	(11 + 10 n)						**Output, Increment, and Repeat** dst ← src Autoincrement src address R ← R − 1 Repeat until R = 0
SIN† **SINB**†	R,src	DA	12	12	12				**Special Input** R ← src
SIND† **SINDB**†	dst,src,R	IR	21	21	21				**Special Input and Decrement** dst ← src Autodecrement dst address R ← R − 1
SINDR† **SINDRB**†	dst,src,R	IR	11 + 10 n)						**Special Input, Decrement, and** Repeat dst ← src Autodecrement dst address R ← R − 1 Repeat until R = 0
SINI† **SINIB**†	dst,src,R	IR	21	21	21				**Special Input and Increment** dst ← src Autoincrement dst address R ← R − 1

*NS = Non-segmented SS = Segmented Short Offset SL = Segmented Long Offset
†Privileged instruction. Executed in system mode only.

INPUT/OUTPUT (Continued)

Mnemonics	Operands	Addr. Modes	Clock Cycles* Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
SINIR† **SINIRB**†	dst,src,R	IR	(11 + 10 n)						**Special Input, Increment, and** Repeat dst ← src Autoincrement dst address R ← R − 1 Repeat until R = 0
SOUT† **SOUTB**†	dst,src	DA	12	12	12				**Special Output** dst ← src
SOUTD† **SOUTDB**†	dst,src,R	IR	21	21	21				**Special Output and Decrement** dst ← src Autodecrement src address R ← R − 1
SOTDR† **SOTDRB**†	dst,src,R	IR	(11 + 10 n)						**Special Output, Decrement, and** Repeat dst ← src Autodecrement src address R ← R − 1 Repeat until R = 0
SOUTI† **SOUTIB**†	dst,src,R	IR	21	21	21				**Special Output and Increment** dst ← src Autoincrement src address R ← R − 1
SOTIR† **SOTIRB**†	dst,src,R	R	(11 + 10 n)						**Special Output, Increment, and** Repeat dst ← src Autoincrement src address R ← R − 1 Repeat until R = 0

CPU CONTROL

COMFLG	flags	—	7	7	7				**Complement Flag** (Any combination of C, Z, S, P/V)
DI†	int	—	7	7	7				**Disable Interrupt** (Any combination of NVI, VI)
EI†	int	—	7	7	7				**Enable Interrupt** (Any combination of NVI, VI)
HALT†	—	—	(8 + 3 n)						**HALT**
LDCTL†	CTLR,src	R	7	7	7				**Load into Control Register** CTLR ← src
LDCTL†	dst,CTLR	R	7	7	7				**Load from Control Register** dst ← CTLR

*NS = Non-segmented SS = Segmented Short Offset SL = Segmented Long Offset
†Privileged instruction Executed in system mode only

CPU CONTROL (Continued)

Mnemonics	Operands	Addr. Modes	Clock Cycles* Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
LDCTLB	FLGR,src	R	7	7	7				**Load into Flag Byte Register** FLGR ← src
LDCTLB	dst,FLGR	R	7	7	7				**Load from Flag Byte Register** dst ← FLGR
LDPS†	src	IR	12	16	16				**Load Program Status** PS ← src
		DA	16	20	22				
		X	17	20	23				
MBIT†	—	—	7	7	7				**Test Multi-Micro Bit** Set S if MI is Low; reset S if MI is High
MREQ†	dst	R	(12 + n)						**Multi-Micro Request**
MRES†	—	—	5	5	5				**Multi-Micro Reset**
MSET†	—	—	5	7	7				**Multi-Micro Set**
NOP	—	—	7	7	7				**No Operation**
RESFLG	flag	—	7	7	7				**Reset Flag** (Any combination of C, Z, S, P/V)
SETFLG	flag	—	7	7	7				**Set Flag** (Any combination of C, Z, S, P/V)

*NS = Non-segmented SS = Segmented Short Offset SL = Segmented Long Offset
†Privileged instruction. Executed in system mode only.

CONDITION CODES

Code	Meaning	Flag Settings	CC Field
F	Always false	—	0000
T	Always true	—	1000
Z	Zero	Z = 1	0110
NZ	Not zero	Z = 0	1110
C	Carry	C = 1	0111
NC	No Carry	C = 0	1111
PL	Plus	S = 0	1101
MI	Minus	S = 1	0101
NE	Not equal	Z = 0	1110
EQ	Equal	Z = 1	0110
OV	Overflow	P/V = 1	0100
NOV	No overflow	P/V = 0	1100
PE	Parity is even	P/V = 1	0100
PO	Parity is odd	P/V = 0	1100
GE	Greater than or equal (signed)	(S XOR P/V) = 0	1001
LT	Less than (signed)	(S XOR P/V) = 1	0001
GT	Greater than (signed)	[Z OR (S XOR P/V)] = 0	1010
LE	Less than or equal (signed)	[Z OR (S XOR P/V)] = 1	0010
UGE	Unsigned greater than or equal	C = 0	1111
ULT	Unsigned less than	C = 1	0111
UGT	Unsigned greater than	[(C = 0) AND (Z = 0)] = 1	1011
ULE	Unsigned less than or equal	(C OR Z) = 1	0011

Note that some condition codes have identical flag settings and binary fields in the instruction:
Z = EQ, NZ = NE, C = ULT, NC = UGE, OV = PE, NOV = PO

STATUS CODE LINES

ST_0-ST_3	Definition
0000	Internal operation
0001	Memory refresh
0010	I/O reference
0011	Special I/O reference (e.g., to an MMU)
0100	Segment trap acknowledge
0101	Non-maskable interrupt acknowledge
0110	Non-vectored interrupt acknowledge
0111	Vectored interrupt acknowledge
1000	Data memory request
1001	Stack memory request
1010	Data memory request (EPU)
1011	Stack memory request (EPU)
1100	Program reference, nth word
1101	Instruction fetch, first word
1110	Extension processor transfer
1111	Reserved

PIN DESCRIPTION

AD_0-AD_{15}. *Address/Data* (inputs/outputs, active High, 3-state). These multiplexed address and data lines are used for I/O and to address memory.

$\overline{AS}$. *Address Strobe* (output, active Low, 3-state). The rising edge of $\overline{AS}$ indicates addresses are valid.

$\overline{BUSACK}$. *Bus Acknowledge* (output active Low). A Low on this line indicates the CPU has relinquished control of the bus.

$\overline{BUSREQ}$. *Bus Request* (input, active Low). This line must be driven Low to request the bus from the CPU.

B/$\overline{W}$. *Byte/Word* (output, Low = Word, 3-state). This signal defines the type of memory reference on the 16-bit address/data bus.

CLK. *System Clock* (input). CLK is a 5V single-phase time-base input.

$\overline{DS}$. *Data Strobe* (output, active Low, 3-state). This line times the data in and out of the CPU.

$\overline{MREQ}$. *Memory Request* (output, active Low, 3-state). A Low on this line indicates that the address/data bus holds a memory address.

$\overline{MI}$, $\overline{MO}$. *Multi-Micro In, Multi-Micro Out* (input and output, active Low). These two lines form a resource-request daisy chain that allows one CPU in a multi-microprocessor system to access a shared resource.

$\overline{NMI}$. *Non-Maskable Interrupt* (edge triggered, input, active Low). A high-to-low transition on $\overline{NMI}$ requests a non-maskable interrupt. The $\overline{NMI}$ interrupt has the highest priority of the three types of interrupts.

N/$\overline{S}$. *Normal/System Mode* (output, Low = System Mode, 3-state). N/$\overline{S}$ indicates the CPU is in the normal or system mode.

$\overline{NVI}$. *Non-Vectored Interrupt* (input, active Low). A Low on this line requests a non-vectored interrupt.

$\overline{RESET}$. *Reset* (input, active Low). A Low on this line resets the CPU.

R/$\overline{W}$. *Read/Write* (output, Low = Write, 3-state). R/$\overline{W}$ indicates that the CPU is reading from or writing to memory or I/O.

$\overline{SEGT}$. *Segment Trap* (input, active Low). The Memory Management Unit interrupts the CPU with a Low on this line when the MMU detects a segmentation trap. Input on Z8001 only.

SN_0-SN_6. *Segment Number* (outputs, active High, 3-state). These lines provide the 7-bit segment number used to address one of 128 segments by the Z8010 memory Management Unit. Output by the Z8001 only.

ST_0-ST_3. *Status* (outputs, active High, 3-state). These lines specify the CPU status (see Status Code Lines).

$\overline{STOP}$. *Stop* (input, active Low). This input can be used to single-step instruction execution.

$\overline{VI}$. *Vectored Interrupt* (input, active Low). A Low on this line requests a vectored interrupt.

$\overline{WAIT}$. *Wait* (input, active Low). This line indicates to the CPU that the memory or I/O device is not ready for data transfer.

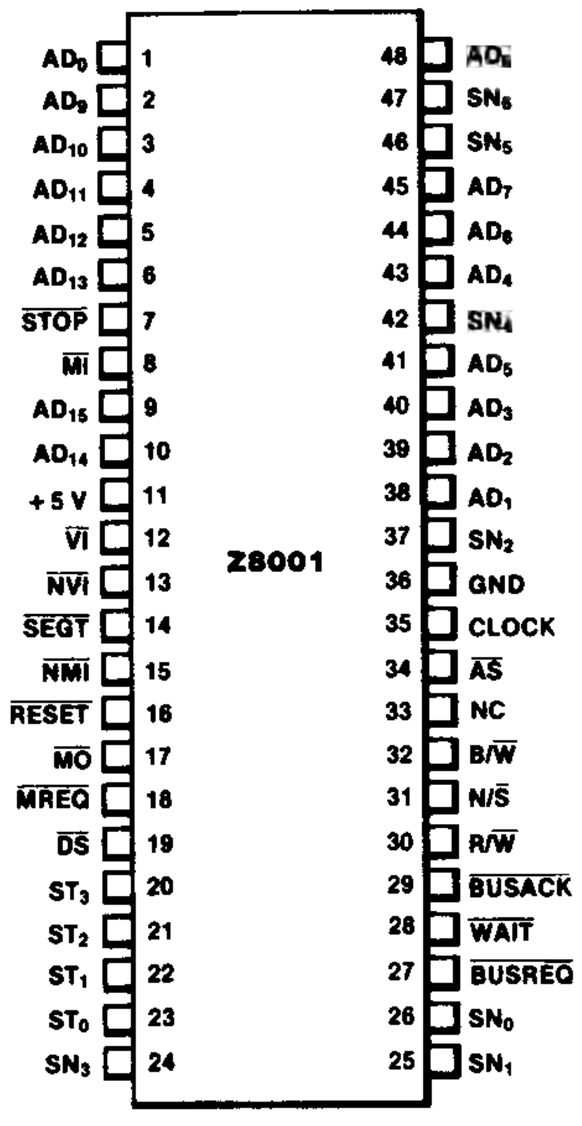


Figure 10a. 48-pin Dual-In-Line Package (DIP), Pin Assignments

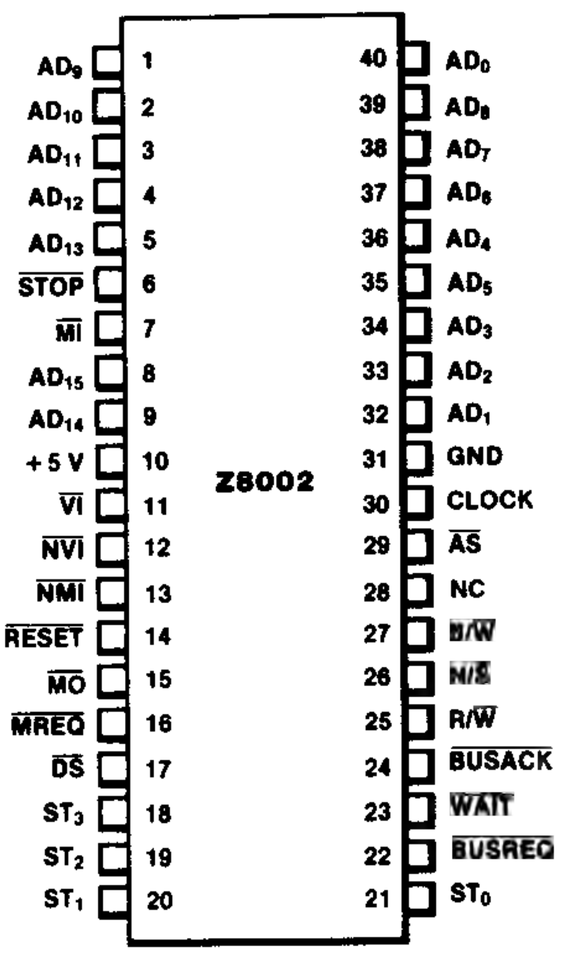
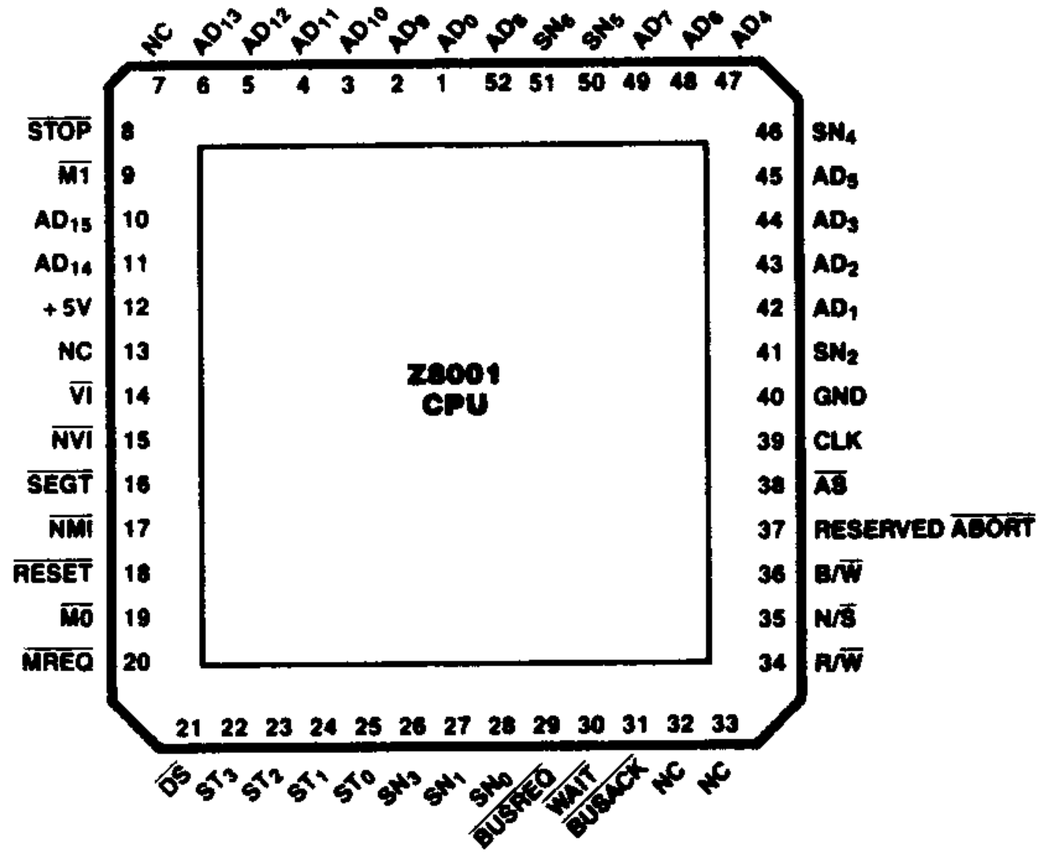


Figure 11a. 40-pin Dual-In-Line Package (DIP), Pin Assignments



NC = No connection

52-pin Chip Carrier, Pin Assignments

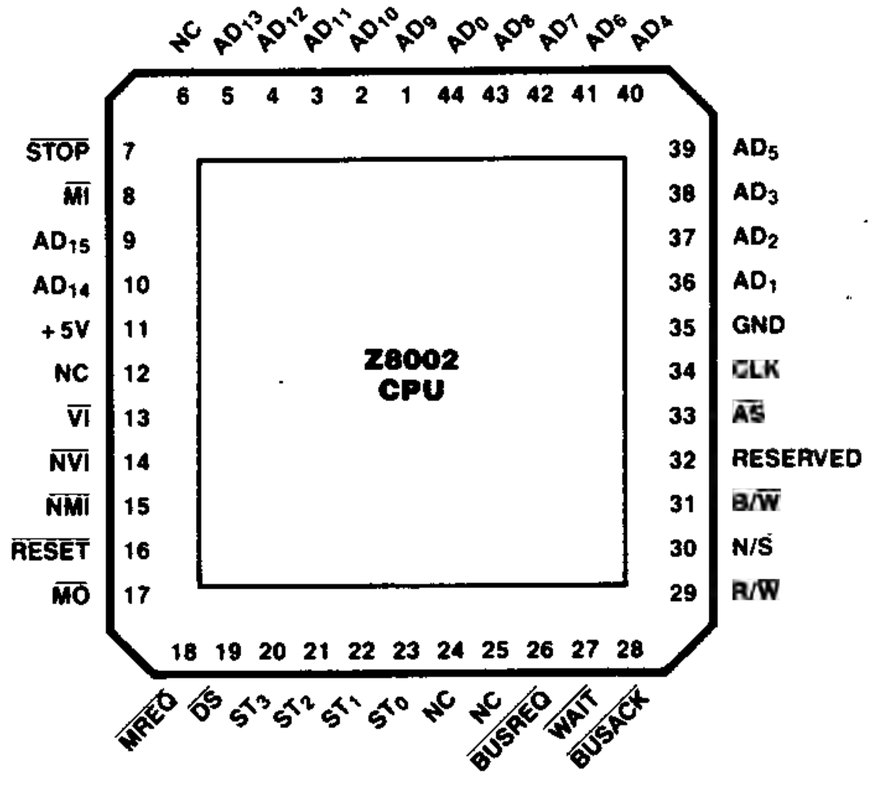


Figure 11b.

44-pin Chip Carrier, Pin Assignments

Z8000 CPU TIMING

The Z8000 CPU executes instructions by stepping through sequences of basic machine cycles, such as memory read or write, I/O device read or write, interrupt acknowledge, and internal execution. Each of these basic cycles requires three to ten clock cycles to execute. Instructions that require more clock cycles to execute are broken up into several machine cycles. Thus no machine cycle is longer than ten clock cycles and fast response to a Bus Request is guaranteed.

The instruction opcode is fetched by a normal memory read operation. A memory refresh cycle can be inserted just after the completion of any first instruction fetch (IF_1) cycle and can also be inserted while the following instructions are being executed: MULT, MULTL, DIV, DIVL, HALT, all Shift instructions, all Block Move instructions, and the Multi-Micro Request instruction (MREQ).

The following timing diagrams show the relative timing relationships of all CPU signals during each of the basic operations. When a machine cycle requires additional clock cycles for CPU internal operation, one to five clock cycles are added. Memory and I/O read and write, as well as interrupt acknowledge cycles, can be extended by activating the $\overline{WAIT}$ input. For exact timing information, refer to the composite timing diagram.

Note that the $\overline{WAIT}$ input is not synchronized in the Z8000 and that the setup and hold times for $\overline{WAIT}$, relative to the clock, must be met. If asynchronous $\overline{WAIT}$ signals are generated, they must be synchronized with the CPU clock before entering the Z8000.

MEMORY READ AND WRITE

Memory read and instruction fetch cycles are identical, except for the status information on the ST_0-ST_3 outputs. During a memory read cycle, a 16-bit address is placed on the AD_0-AD_{15} outputs early in the first clock period, as shown in Figure 12. In the Z8001, the 7-bit segment number is output on SN_0-SN_6 one clock period earlier than the 16-bit address offset.

A valid address is indicated by the rising edge of Address Strobe. Status and mode information become valid early in the memory access cycle and remain stable throughout. The state of the $\overline{WAIT}$ input is sampled in the middle of the second clock cycle by the falling edge of Clock. If $\overline{WAIT}$ is Low, an additional clock period is added between T_2 and T_3. $\overline{WAIT}$ is sampled again in the middle of this wait cycle, and additional wait states can be inserted: this allows interfacing slow memories. No control outputs change during wait states.

Although Z8000 memory is word organized, memory is addressed as bytes. All instructions are word-aligned, using even addresses. Within a 16-bit word, the most significant byte (D_8-D_{15}) is addressed by the low-order address (A_0 = Low), and the least significant byte (D_0-D_7) is addressed by the high-order address (A_0 = High).

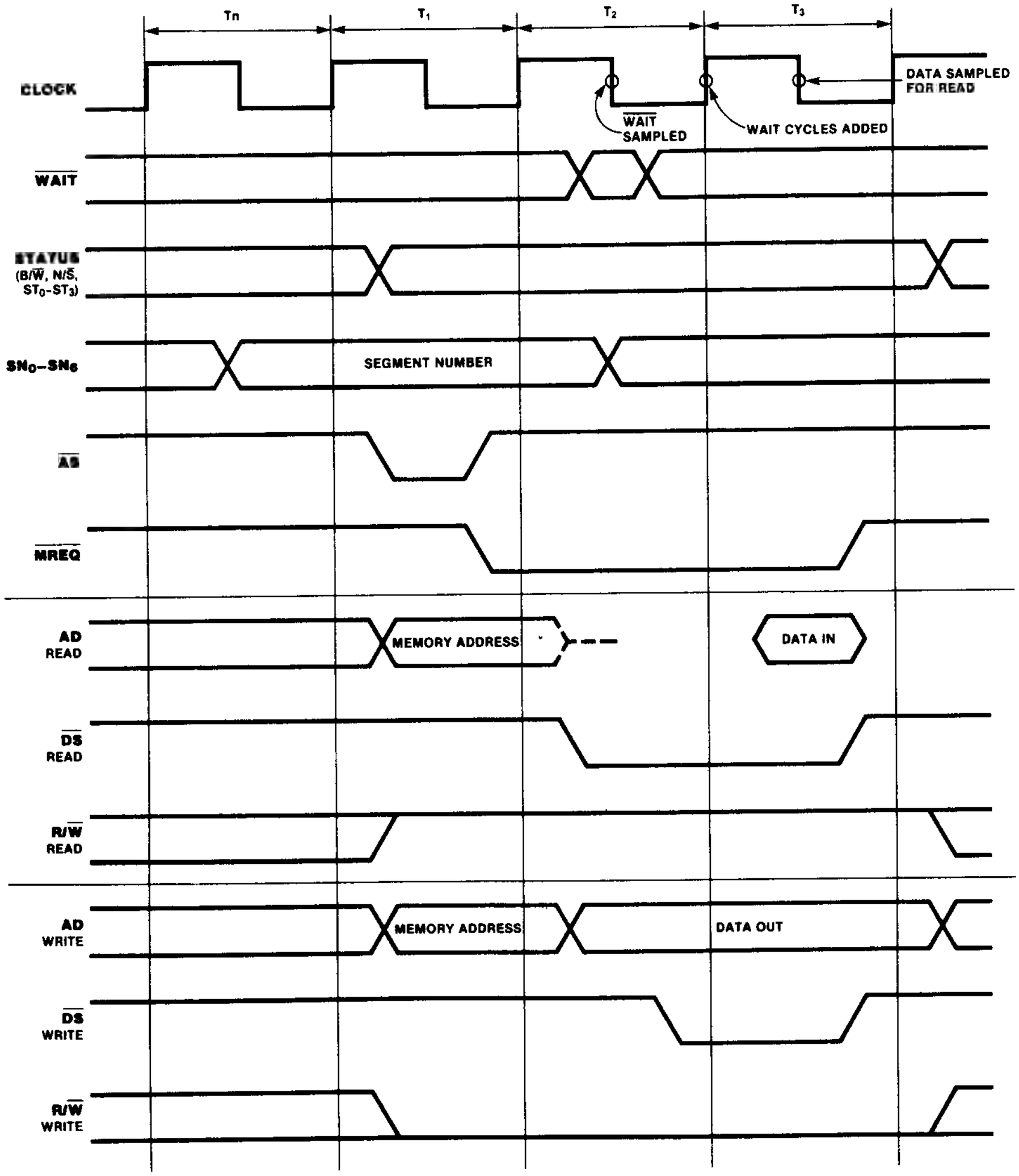


Figure 12. Memory Read and Write Timing

INPUT/OUTPUT

I/O timing is similar to memory read/write timing, except that one wait state is automatically (T_{WA}) inserted between T_2 and T_3 (Figure 13). Both the segmented Z8001/Z8005 and the nonsegmented Z8002 use 16-bit I/O addresses.

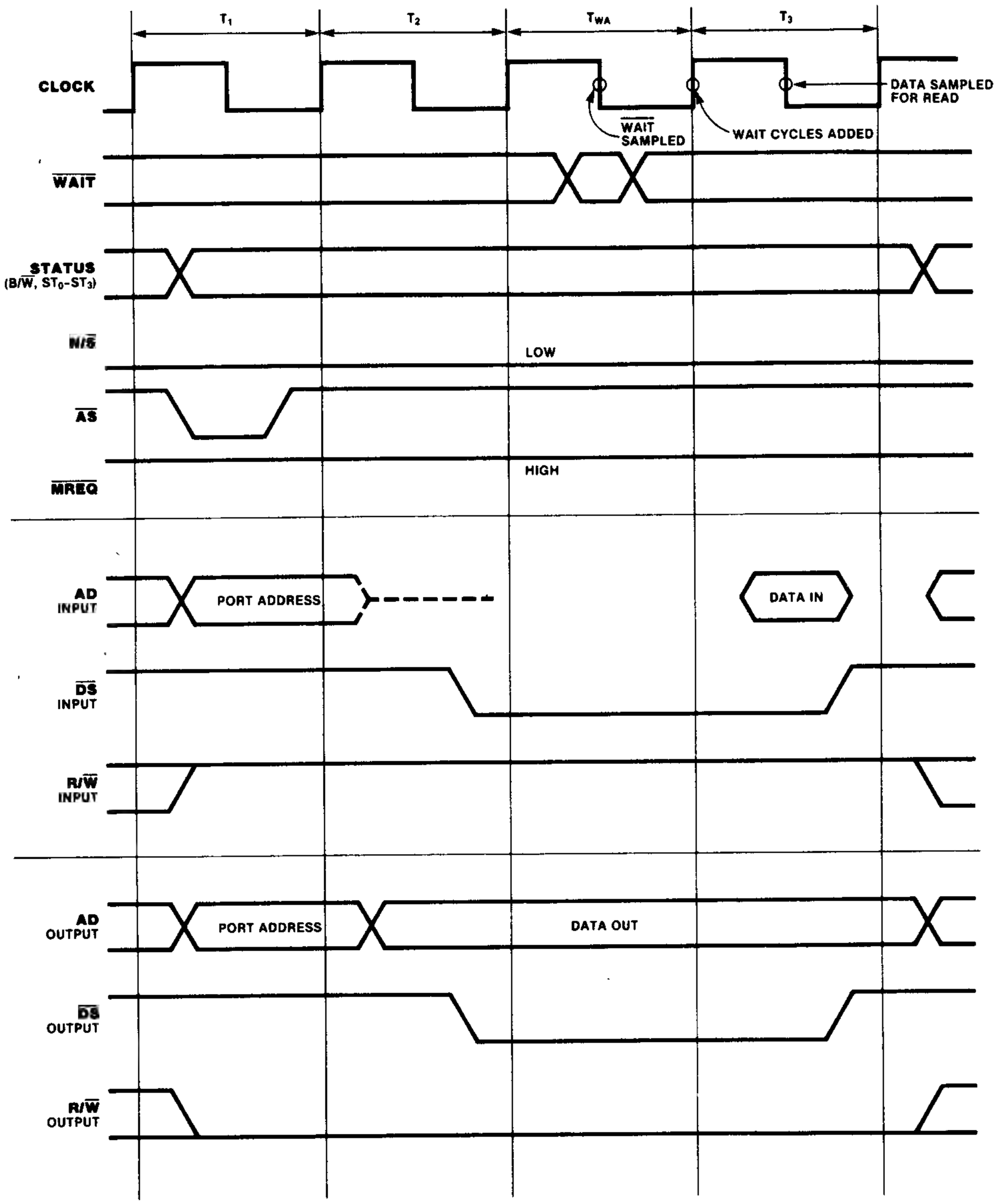


Figure 13. Input/Output Timing

INTERRUPT AND SEGMENT TRAP REQUEST AND ACKNOWLEDGE

The Z8000 CPU recognizes three interrupt inputs (non-maskable, vectored, and nonvectored) and a segmentation trap input. Any High-to-Low transition on the $\overline{NMI}$ input is asynchronously edge detected and sets the internal $\overline{NMI}$ latch. The $\overline{VI}$, $\overline{NVI}$, and $\overline{SEGT}$ inputs, as well as the state of the internal NMI latch, are sampled at the end of T_2 in the last machine cycle of any instruction.

In response to an interrupt or trap, the subsequent IF_1 cycle is exercised, but ignored. The internal state of the CPU is not altered and the instruction will be refetched and executed after the return from the interrupt routine. The program counter is not updated, but the system stack pointer is decremented in preparation for pushing starting information onto the system stack.

The next machine cycle is the interrupt acknowledge cycle. This cycle has five automatic wait states, with additional wait states possible, as shown in Figure 14.

After the last wait state, the CPU reads the information on AD_0-AD_{15} and temporarily stores it, to be saved on the stack later in the acknowledge sequence. This word identifies the source of the interrupt or trap. For the nonvectored and nonmaskable interrupts, all 16 bits can represent peripheral device status information. For the vectored interrupt, the low byte is the jump vector, and the high byte can be extra user status. For the segmentation trap, the *high* byte is the Memory Management Unit identifier and the *low* byte is undefined.

After the acknowledge cycle, the N/$\overline{S}$ output indicates the automatic change to system mode.

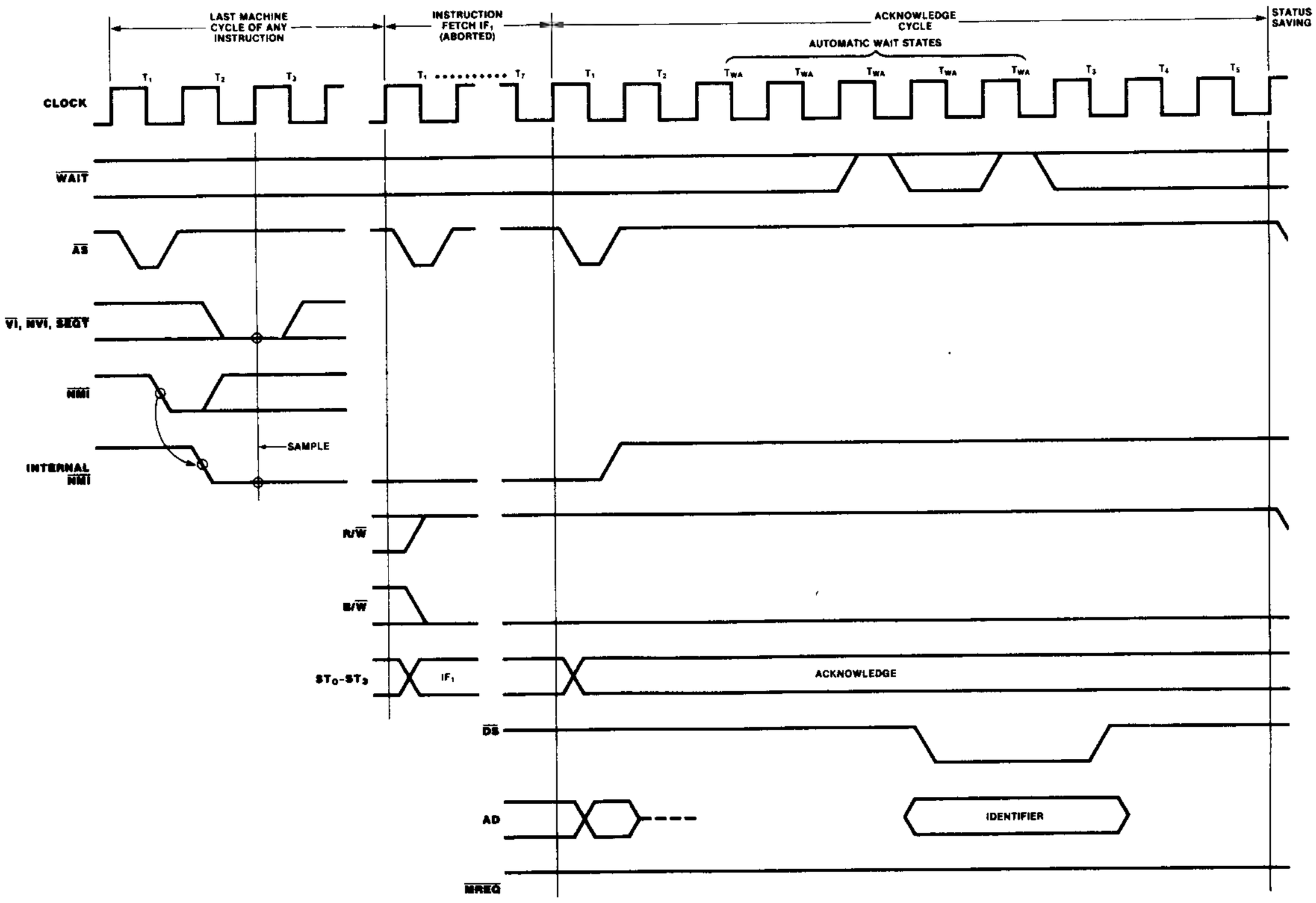


Figure 14. Interrupt and Segment Trap Request/Acknowledge Timing

STATUS SAVING SEQUENCE

The machine cycles, following the interrupt acknowledge or segmentation trap acknowledge cycle, push the old status information on the system stack in the following order: the 16-bit program counter; the 7-bit segment number (Z8001/Z8005 only); the flag control word; and finally the interrupt/trap identifier. Subsequent machine cycles fetch the new program status from the program status area, and then branch to the interrupt/trap service routine.

BUS REQUEST ACKNOWLEDGE TIMING

A Low on the $\overline{\text{BUSREQ}}$ input indicates to the CPU that another device is requesting the Address/Data and control buses. The asynchronous $\overline{\text{BUSREQ}}$ input is synchronized at the beginning of any machine cycle (Figure 15). $\overline{\text{BUSREQ}}$ takes priority over WAIT. If $\overline{\text{BUSREQ}}$ is Low, an internal synchronous $\overline{\text{BUSREQ}}$ signal is generated, which—after completion of the current machine cycle—causes the $\overline{\text{BUSACK}}$ output to go Low and all bus outputs to go into the high-impedance state. The requesting device—typically a DMA—can then control the bus.

When $\overline{\text{BUSREQ}}$ is released, it is synchronized with the rising clock edge; the $\overline{\text{BUSACK}}$ output goes High one clock period later, indicating that the CPU will again take control of the bus.

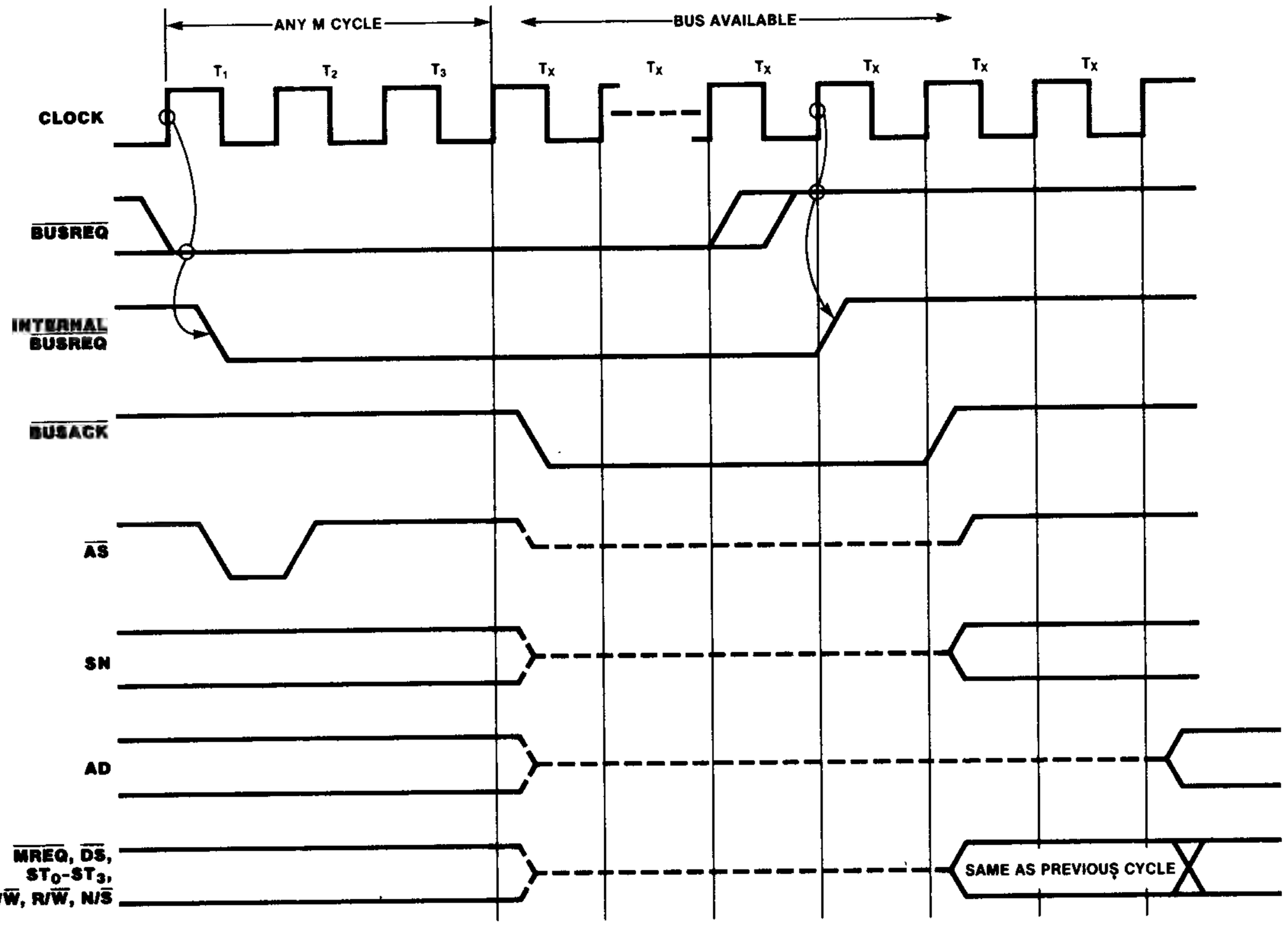


Figure 15. Bus Request/Acknowledge Timing

STOP

The STOP input is sampled by the last falling clock edge immediately preceding any IF_1 cycle (Figure 16) and before the second word of an EPA instruction is fetched. If $\overline{STOP}$ is found Low during the IF_1 cycle, a stream of memory refresh cycles is inserted after T_3, again sampling the $\overline{STOP}$ input on each falling clock edge in the middle of the T_3 states. During the EPA instruction, both EPA instruction words are fetched but any data transfer or subsequent instruction fetch is postponed until $\overline{STOP}$ is sampled High. This refresh operation does not use the refresh prescaler or its divide-by-four clock prescaler; rather, it double-increments the refresh counter every three clock cycles. When $\overline{STOP}$ is found High again, the next refresh cycle is completed, any remaining T states of the IF_1 cycle are then executed, and the CPU continues its operation.

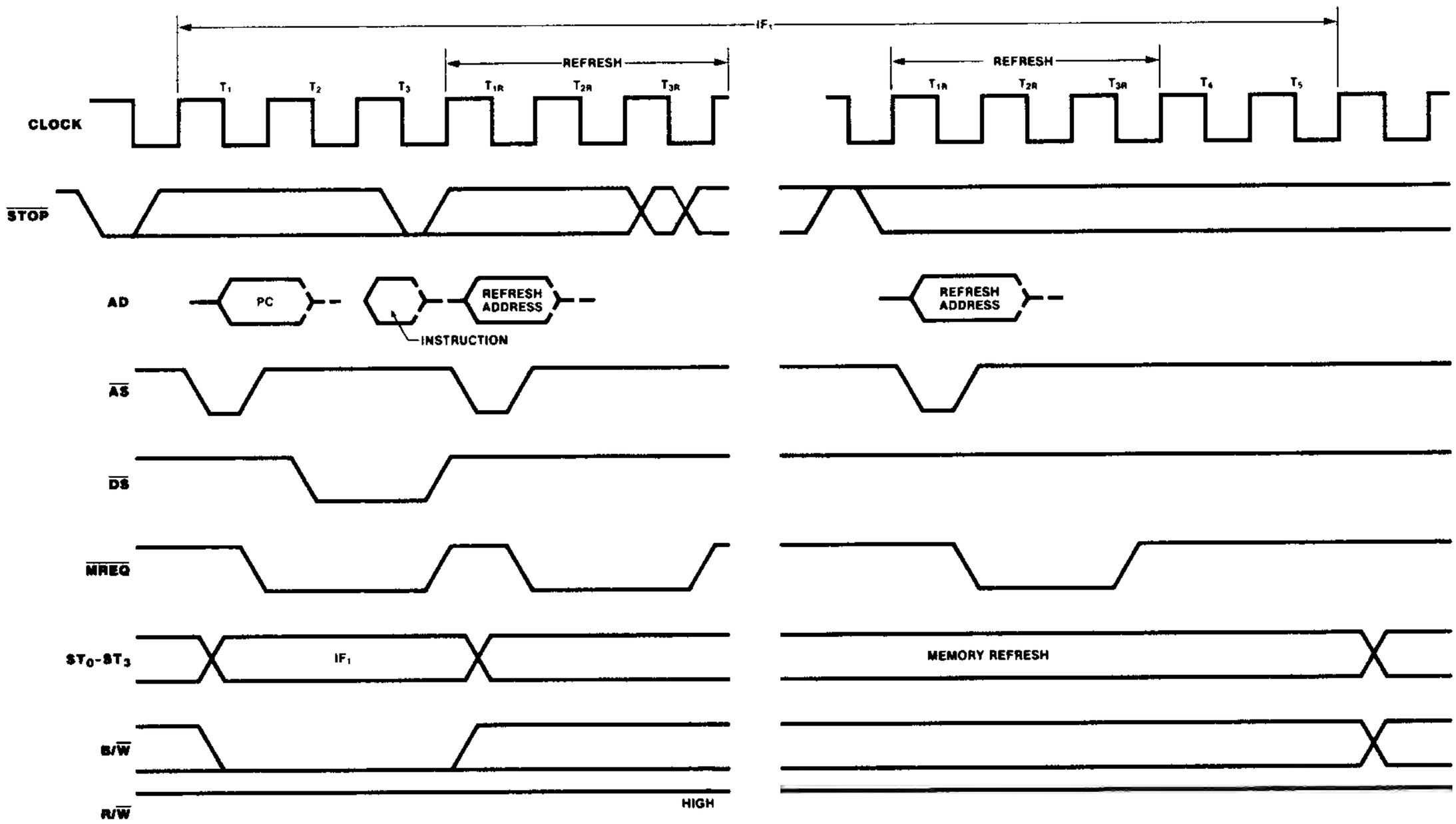


Figure 16. Stop Timing

INTERNAL OPERATION

Certain extended instructions, such as Multiply and Divide, and some special instructions need additional time for the execution of internal operations. In these cases, the CPU goes through a sequence of internal operation machine cycles, each of which is three to eight clock cycles long (Figure 17). This allows fast response to Bus Request and Refresh Request, because bus request or refresh cycles can be inserted at the end of any internal machine cycle.

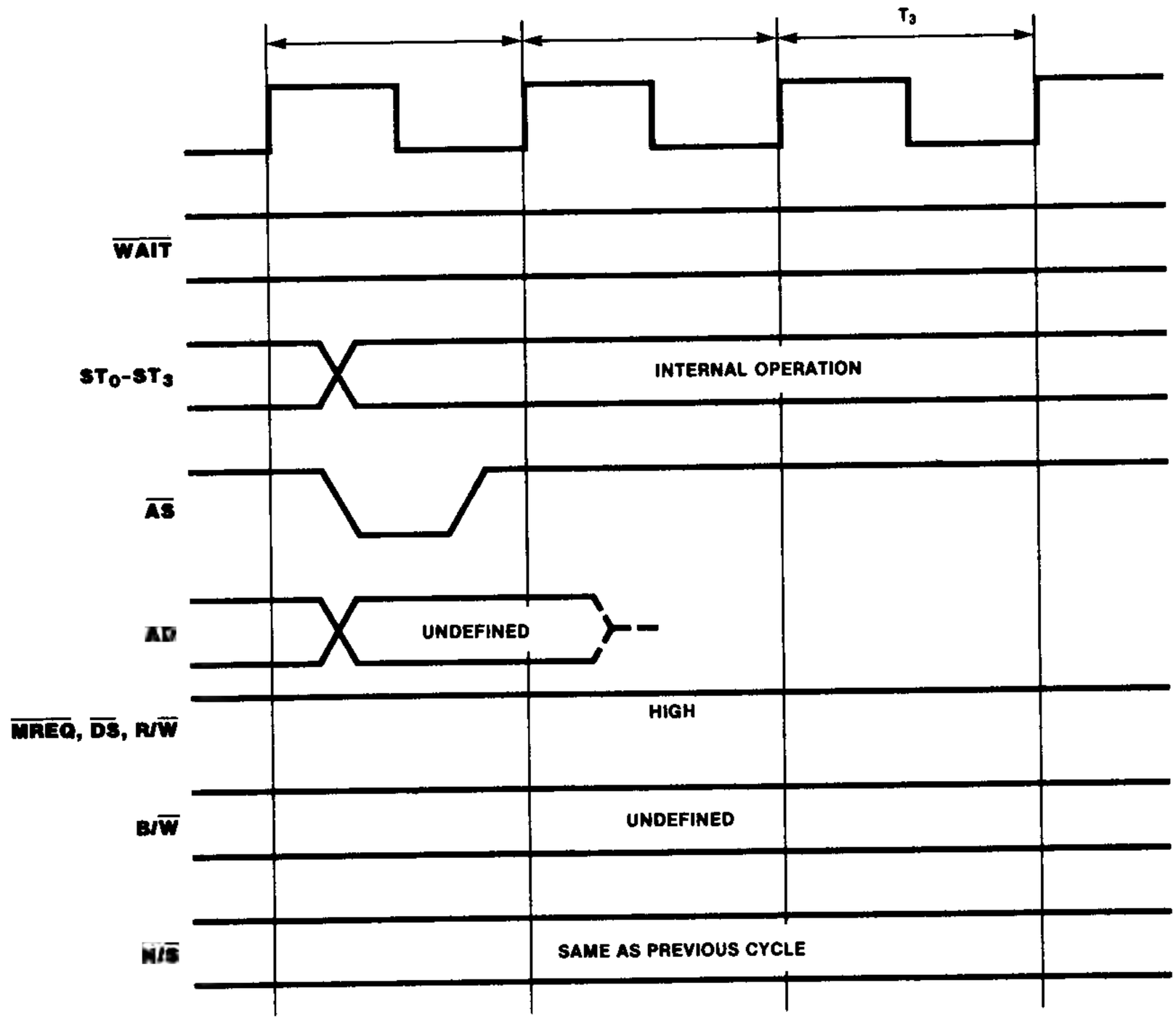


Figure 17. Internal Operation Timing

HALT

A HALT instruction executes an unlimited number of 3-cycle internal operations, interspersed with memory refresh cycles whenever requested. An interrupt, segmentation trap, or reset are the only exits from a HALT instruction.

The CPU samples the $\overline{VI}$, $\overline{NVI}$, $\overline{NMI}$, and $\overline{SEGT}$ inputs at the beginning of every T_3 cycle. If an input is found active during two consecutive samples, the subsequent IF_1 cycle is exercised, but ignored, and the normal interrupt acknowledge cycle is started.

MEMORY REFRESH

When the 6-bit prescaler in the refresh counter has been decremented to zero, a refresh cycle consisting of three T-states is started as soon as possible (that is, after the next IF_1 cycle or Internal Operation cycle).

The 9-bit refresh counter value is put on the low-order side of the address bus (AD_0-AD_8); AD_9-AD_{15} are undefined (Figure 18). Since the memory is word-organized, A_0 is always Low during refresh and the refresh counter is always incremented by two, thus stepping through 256 consecutive refresh addresses on AD_1-AD_8. Unless disabled, the presettable prescaler runs continuously and the delay in starting a refresh cycle is therefore not cumulative.

While the $\overline{STOP}$ input is Low, a continuous stream of memory refresh cycles, each three T-states long, is executed without using the refresh prescaler.

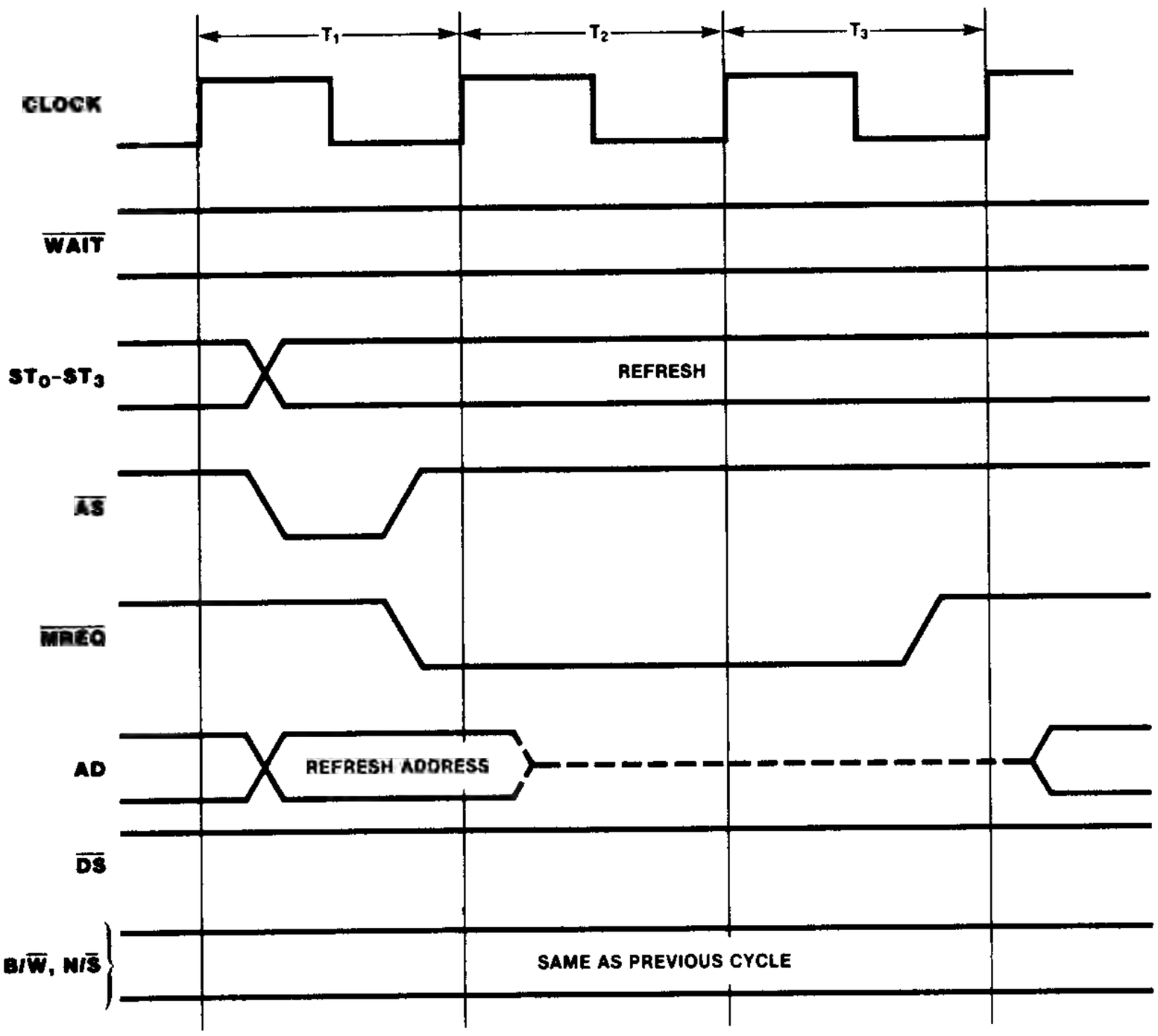


Figure 18. Memory Refresh Timing

RESET

A Low on the $\overline{RESET}$ input causes the following results within five clock cycles (Figure 19):

- AD_0-AD_{15} are 3-stated
- $\overline{AS}$, $\overline{DS}$, $\overline{MREQ}$, ST_0-ST_3, $\overline{BUSACK}$, and $\overline{MO}$ are forced High
- SN_0-SN_6 are forced Low
- Refresh is disabled
- R/$\overline{W}$, B/$\overline{W}$, and N/$\overline{S}$ are not affected

When $\overline{RESET}$ has been High for three clock periods, three consecutive memory read cycles are executed in the system mode for the Z8001. The Z8002 has two consecutive read cycles. In the Z8001, the first cycle reads the flag and control word from location 0002, the next reads the 7-bit program counter segment number from location 0004, the next reads the 16-bit PC offset from location 0006, and the following IF_1 cycle starts the program. In the Z8002, the first cycle reads the flag and control word from location 0002, the next reads the PC from location 0004, and the following IF_1 cycle starts the program.

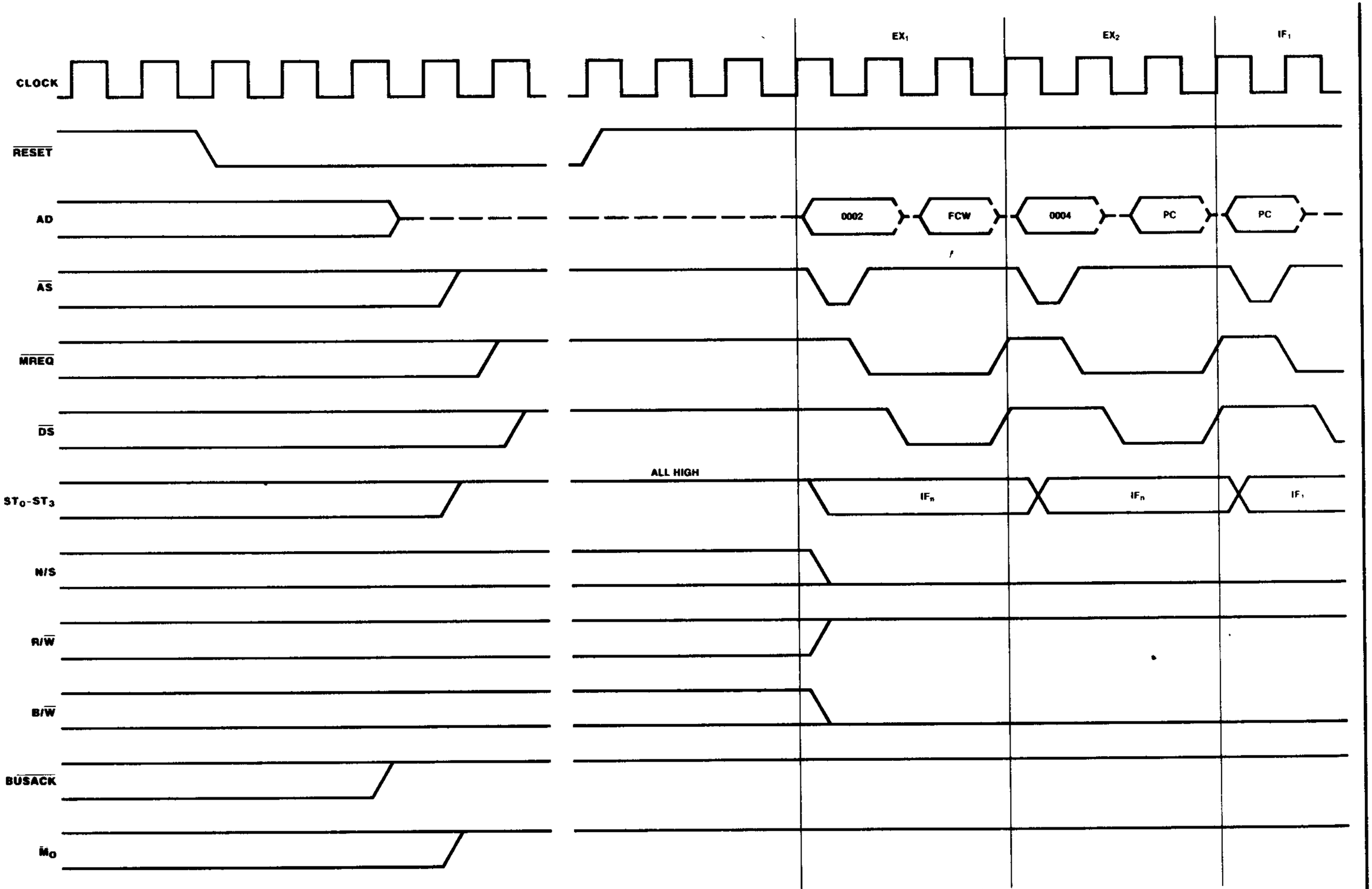
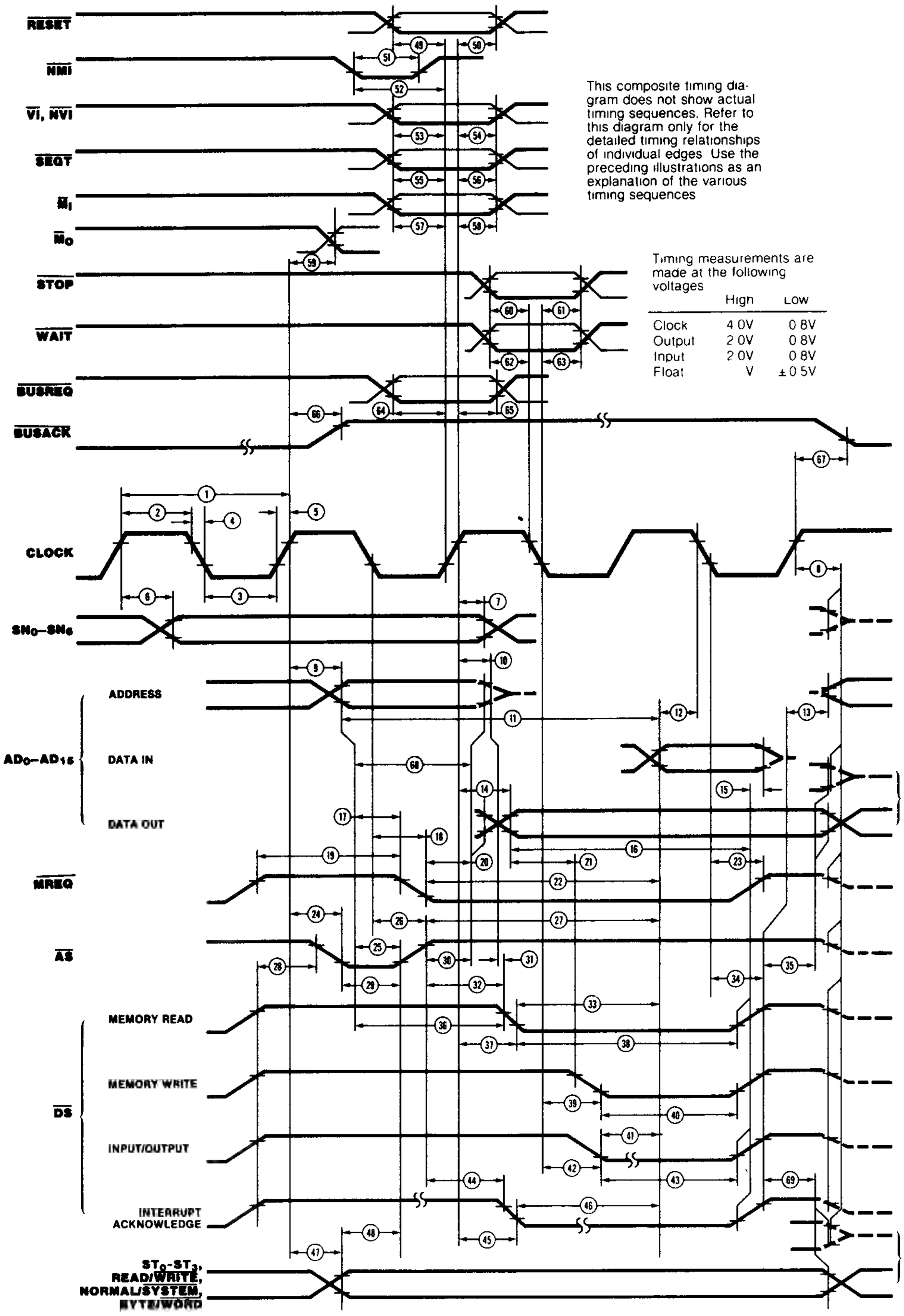


Figure 19. Reset Timing

COMPOSITE AC TIMING DIAGRAM



This composite timing diagram does not show actual timing sequences. Refer to this diagram only for the detailed timing relationships of individual edges Use the preceding illustrations as an explanation of the various timing sequences

Timing measurements are made at the following voltages

	High	Low
Clock	4 0V	0 8V
Output	2 0V	0 8V
Input	2 0V	0 8V
Float	V	±0 5V

AC CHARACTERISTICS†

Number	Symbol	Parameter	Z8001/2 4 MHz Min	Z8001/2 4 MHz Max	Z8001/2 6 MHz Min	Z8001/2 6 MHz Max	Z8001/2 10 MHz Min	Z8001/2 10 MHz Max
1	TcC	Clock Cycle Time	250	2000	165	2000	100	2000
2	TwCh	Clock Width (High)	105	1895	70	1930	40	1960
3	TwCl	Clock Width (Low)	105	1895	70	1930	40	1960
4	TfC	Clock Fall Time		20		10		10
5	TrC	Clock Rise Time		20		15		10
6	TdC(SNv)	Clock ↑ to Segment Number Valid (50 pf load)		130		110		90
7	TdC(SNn)	Clock ↑ to Segment Number Not Valid	20		10		0	
8	TdC(Bz)	Clock ↑ to Bus Float		65		55		50
9	TdC(A)	Clock ↑ to Address Valid		100		75		55
10	TdC(Az)	Clock ↑ to Address Float		65		55		50
11	TdA(DR)	Address Valid to Read Data Required Valid		475*		305*		180*
12	TsDR(C)	Read Data to Clock ↓ Setup time	30		20		10	
13	TdDS(A)	$\overline{DS}$ ↑ to Address Active	80*		45*		20*	
14	TdC(DW)	Clock ↑ to Write Data Valid		100		75		60
15	ThDR(DS)	Read Data to $\overline{DS}$ ↑ Hold Time	0		0		0	
16	TdDW(DS)	Write Data Valid to $\overline{DS}$ ↑ Delay	295*		195*		110*	
17	TdA(MR)	Address Valid to $\overline{MREQ}$ ↓ Delay	55*		35*		20*	
18	TdC(MR)	Clock ↓ to $\overline{MREQ}$ ↓ Delay		80		70		50
19	TwMRh	$\overline{MREQ}$ Width (High)	210*		135*		80*	
20	TdMR(A)	$\overline{MREQ}$ ↓ to Address Not Active	70*		35*		20*	
21	TdDW(DSW)	Write Data Valid to $\overline{DS}$ ↓ (Write) Delay	55*		35*		15*	
22	TdMR(DR)	$\overline{MREQ}$ ↓ to Read Data Required Valid		370*		230*		140*
23	TdC(MR)	Clock ↓ $\overline{MREQ}$ ↑ Delay		80		60		50
24	TdC(ASf)	Clock ↑ to $\overline{AS}$ ↓ Delay		80		60		45
25	TdA(AS)	Address Valid to $\overline{AS}$ ↑ Delay	55*		35*		20*	
26	TdC(ASr)	Clock ↓ to $\overline{AS}$ ↑ Delay		90		80		45
27	TdAS(DR)	$\overline{AS}$ ↑ to Read Data Required Valid		360*		220*		140*
28	TdDS(AS)	$\overline{DS}$ ↑ to $\overline{AS}$ ↓ Delay	70*		35*		15*	
29	TwAS	$\overline{AS}$ Width (Low)	85*		55*		30*	
30	TdAS(A)	$\overline{AS}$ ↑ to Address Not Active Delay	70*		45*		20*	
31	TdAz(DSR)	Address Float to $\overline{DS}$ (Read) ↓ Delay	0		0		0	
32	TdAS(DSR)	$\overline{AS}$ ↑ to $\overline{DS}$ (Read) ↓ Delay	80*		55*		30*	
33	TdDSR(DR)	$\overline{DS}$ (Read) ↓ to Read Data Required Valid		205*		130*		70*
34	TdC(DSr)	Clock ↓ to $\overline{DS}$ ↑ Delay		70		65		50
35	TdDS(DW)	$\overline{DS}$ ↑ to Write Data Not Valid	75*		45*		25*	
36	TdA(DSR)	Address Valid to $\overline{DS}$ (Read) ↓ Delay	180*		110*		65*	
37	TdC(DSR)	Clock ↑ to $\overline{DS}$ (Read) ↓ Delay		120		85		65
38	TwDSR	$\overline{DS}$ (Read) Width (Low)	275*		185*		110*	
39	TdC(DSW)	Clock ↓ to $\overline{DS}$ (Write) ↓ Delay		95		80		65
40	TwDSW	$\overline{DS}$ (Write) Width (Low)	185*		110*		75*	

*Clock-cycle time-dependent characteristics. See Footnotes to AC Characteristics.
†Units in nanoseconds (ns).

AC CHARACTERISTICS† (Continued)

Number	Symbol	Parameter	Z8001/2 4 MHz Min	Z8001/2 4 MHz Max	Z8001/2 6 MHz Min	Z8001/2 6 MHz Max	Z8001/2 10 MHz Min	Z8001/2 10 MHz Max
41	TdDSI(DR)	$\overline{DS}$ (I/O) ↓ to Read Data Required Valid		330*		210*		120*
42	TdC(DSf)	Clock ↓ to $\overline{DS}$ (I/O) ↓ Delay		120		90		65
43	TwDS	$\overline{DS}$ (I/O) Width (Low)	410*		255*		160*	
44	TdAS(DSA)	$\overline{AS}$ ↑ to $\overline{DS}$ (Acknowledge) ↓ Delay	1065*		690*		410*	
45	TdC(DSA)	Clock ↑ to $\overline{DS}$ (Acknowledge) ↓ Delay		120		85		70
46	TdDSA(DR)	$\overline{DS}$ (Acknowledge) ↓ to Read Data Required Delay		455*		295*		165*
47	TdC(S)	Clock ↑ to Status Valid Delay		110		85		65
48	TdS(AS)	Status Valid to $\overline{AS}$ ↑ Delay	50*		30*		20*	
49	TsR(C)	$\overline{RESET}$ to Clock ↑ Setup Time	180		70		50	
50	ThR(C)	$\overline{RESET}$ to Clock ↑ Hold Time	0		0		0	
51	TwNMI	$\overline{NMI}$ Width (Low)	100		70		50	
52	TsNMI(C)	$\overline{NMI}$ to Clock ↑ Setup Time	140		70		50	
53	TsVI(C)	$\overline{VI}$, $\overline{NVI}$ to Clock ↑ Setup Time	110		50		40	
54	ThVI(C)	$\overline{VI}$, $\overline{NVI}$ to Clock ↑ Hold Time	20		20		10	
55	TsSGT(C)	$\overline{SEGT}$ to Clock ↑ Setup Time	70		55		40	
56	ThSGT(C)	$\overline{SEGT}$ to Clock ↑ Hold Time	0		0		0	
57	TsMI(C)	$\overline{MI}$ to Clock ↑ Setup Time	180		140		80	
58	ThMI(C)	$\overline{MI}$ to Clock ↑ Hold Time	0		0		0	
59	TdC(MO)	Clock ↑ to $\overline{MO}$ Delay		120		85		80
60	TsSTP(C)	$\overline{STOP}$ to Clock ↓ Setup Time	140		100		50	
61	ThSTP(C)	$\overline{STOP}$ to Clock ↓ Hold Time	0		0		0	
62	TsW(C)	$\overline{WAIT}$ to Clock ↓ Setup Time	50		30		20	
63	ThW(C)	$\overline{WAIT}$ to Clock ↓ Hold Time	10		10		5	
64	TsBRQ(C)	$\overline{BUSREQ}$ to Clock ↑ Setup Time	90		80		60	
65	ThBRQ(C)	$\overline{BUSREQ}$ to Clock ↑ Hold Time	10		10		5	
66	TdC(BAKr)	Clock ↑ to $\overline{BUSACK}$ ↑ Delay		100		75		65
67	TdC(BAKf)	Clock ↑ to $\overline{BUSACK}$ ↓ Delay		100		75		65
68	TwA	Address Valid Width	150*		95*		50*	
69	TdDS(S)	$\overline{DS}$ ↑ to STATUS Not Valid	80*		55*		30*	

*Clock-cycle time-dependent characteristics. See Footnotes to AC Characteristics.
†Units in nanoseconds (ns).

IMAGE UNAVAILABLE

ABSOLUTE MAXIMUM RATINGS

Voltages on all pins with respect to GND . . . . . −0.3V to +7.0V
Operating Ambient Temperature . . . . . See Ordering Information
Storage Temperature . . . . . −65°C to +150°C

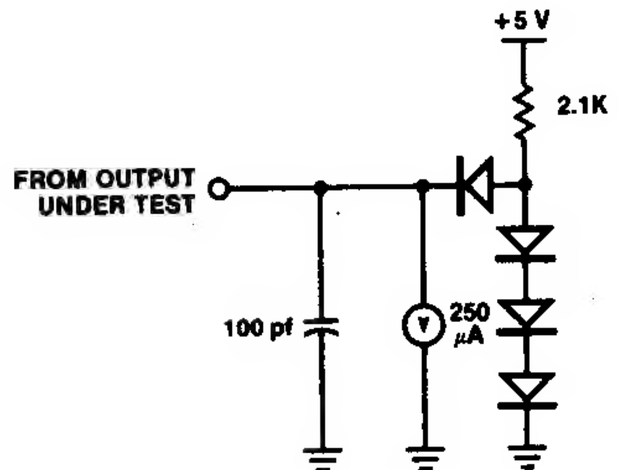
Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The DC characteristics below apply for the following test conditions, unless otherwise noted. All voltages are referenced to GND (0V). Positive current flows into the referenced pin.

Available operating temperature ranges are:

- S = 0°C to +70°C, $+4.75V \leq V_{CC} \leq +5.25V$
- **E = -40°C to +100°C, $+4.75V \leq V_{CC} \leq +5.25V$**



All ac parameters assume a total load capacitance (including parasitic capacitances) or 100 pf max, except for parameter 6 (50 pf max). Timing references between two output signals assume a load difference of 50 pf max.

The Ordering Information section lists package temperature ranges and product numbers.

DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	$V_{CC}-0.4$	$V_{CC}+0.3$	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	−0.3	0.45	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	2.0	$V_{CC}+0.3$	V	
$V_{IH\ RESET}$	Input High Voltage on $\overline{RESET}$ pin	2.4	$V_{CC}+0.3$	V	
$V_{IH\ NMI}$	Input High Voltage on NMI pin	2.4	$V_{CC}+0.3$	V	
V_{IL}	Input Low Voltage	−0.3	0.8	V	
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -250\ \mu A$
V_{OL}	Output Low Voltage		0.4	V	$I_{OL} = +2.0$ mA
I_{IL}	Input Leakage		±10	µA	$0.4 \leq V_{IN} \leq +2.4V$
$I_{IL\ SEGT}$	Input Leakage on $\overline{SEGT}$ pin	−100	100	µA	
I_{OL}	Output Leakage		±10	µA	$0.4 \leq V_{IN} \leq +2.4V$
I_{CC}	V_{CC} Power Supply Current		300	mA	4 MHz and 6 MHz commercial
			400	mA	Extended temperature range
			400	mA	10 MHz speed range